ARDUINO

ขั้นพื้นฐาน สำหรับผู้เริ่มต้น 1

โดยผู้เชี่ยวชาญทั้งด้านฮาร์ดแวร์และซอฟต์แวร์

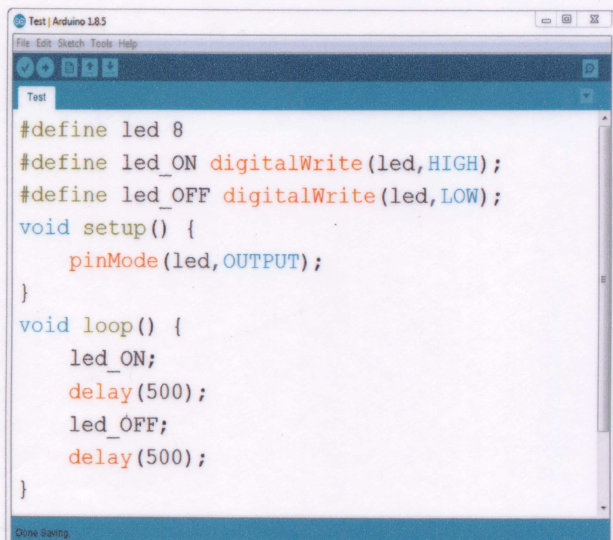


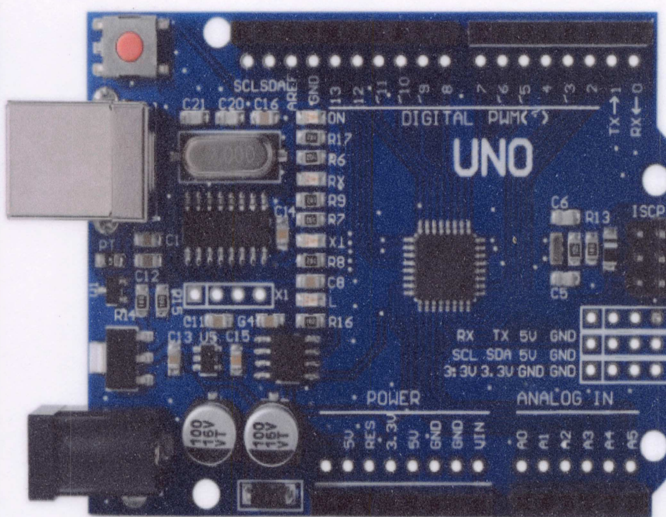
UNO R3

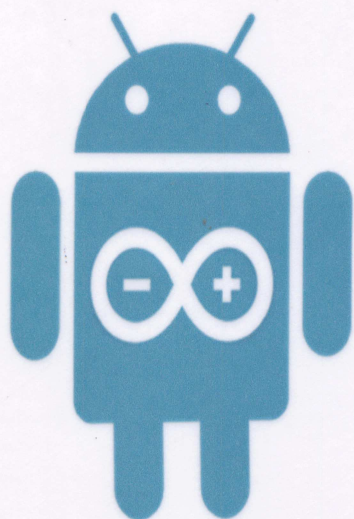
การเขียนโปรแกรม

ด้วยภาษา C++

บน Arduino IDE







เนื้อหาอ่านเข้าใจง่าย

พร้อมตัวอย่างให้ทดลองเพิ่มความเข้าใจ

เหมาะสำหรับผู้เริ่มต้น

และผู้ที่ต้องการศึกษาด้วยตนเอง



มานพ ปักษี

ISBN 978-616-485-872-5

ARDUINO

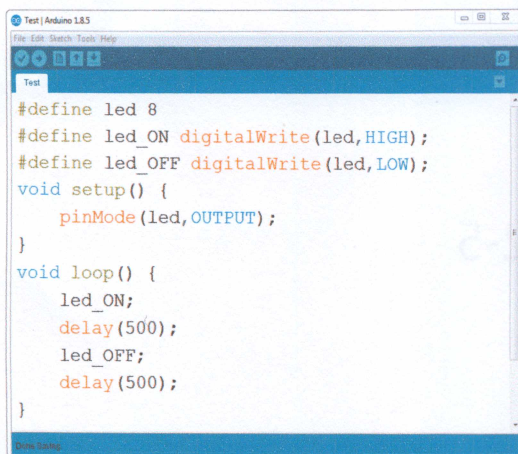
ขั้นพื้นฐาน สำหรับผู้เริ่มต้น 1

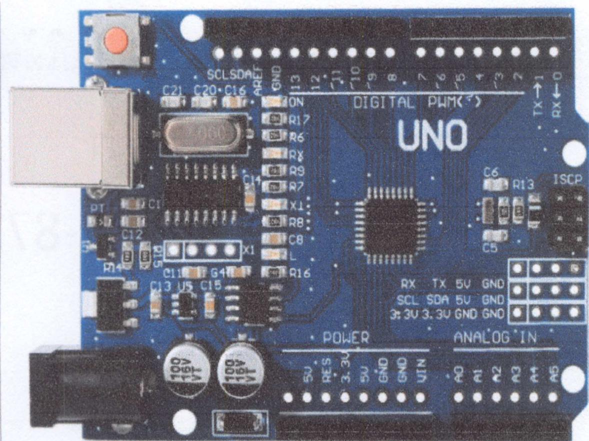
โดยผู้เชี่ยวชาญทั้งด้านฮาร์ดแวร์และซอฟต์แวร์

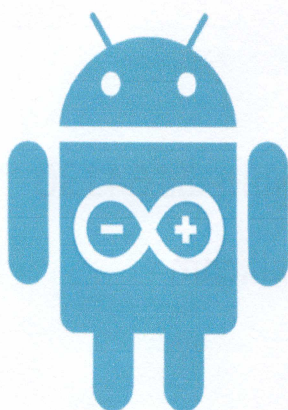


UNO R3

การเขียนโปรแกรม
ด้วยภาษา C++
บน Arduino IDE







เนื้อหาอ่านเข้าใจง่าย

พร้อมตัวอย่างให้ทดลองเพิ่มความเข้าใจ

เหมาะสำหรับผู้เริ่มต้น

และผู้ที่ต้องการศึกษาด้วยตนเอง

ผู้แต่ง/ผู้เขียน มานพ ปักษี

ราคา **350** บาท

เว้นแต่จะได้รับอนุญาตเป็นลายลักษณ์อักษรจากผู้เขียนเท่านั้น

พิมพ์ครั้งที่ จัดพิมพ์ตามสั่ง

จำนวนหน้า 195 หน้า ไม่รวม สารบัญ และ คำนำ

เล่มนี้เป็นงานพิมพ์ด้วยเครื่องอิงค์เจท เข้าเล่มด้วยกาว (งานแฮนด์เมด)

[illegible] ควรเก็บไว้ที่ที่มีอุณหภูมิไม่ร้อน และ ห้ามชื้น

ข้อควรระวัง ไม่ควรให้โดนน้ำ

ISBN 978-616-485-872-5

จัดพิมพ์โดย มานพ ปักษี

จัดจำหน่ายโดย มานพ ปักษี

ร้าน ออนิทีเทค โทรศัพท์ 093-9749793

ที่อยู่ 179 หมู่ 13 ต.สารจิตร อ.ศรีสัชนาลัย จ.สุโขทัย รหัสไปรษณีย์ 64130

nopelectronic@gmail.com / ornittech@gmail.com

สารบัญ

คำนำ

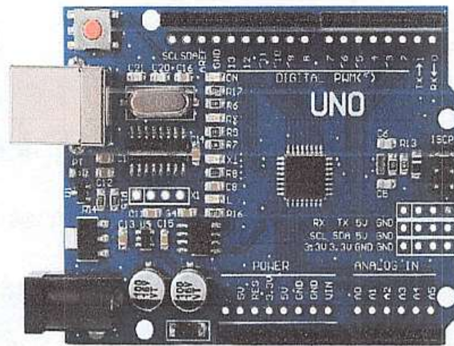
ช่วงเวลาที่ผ่านไปเพียงไม่กี่สิบปีมานี้ อุปกรณ์เครื่องใช้สิ่งอำนวยความสะดวกต่าง ๆ ที่มนุษย์ได้สร้างสรรค์ขึ้นมา เพื่ออำนวยความสะดวกในการดำรงชีวิตนั้นได้พัฒนาไปมาก หากมองย้อนกลับไปเมื่อหลายสิบปีก่อนนั้น อุปกรณ์เครื่องใช้ไฟฟ้าอิเล็กทรอนิกส์ที่มีใช้กันในบ้านเรือนทั่วไป ยังไม่ได้มีความทันสมัยมากนัก การที่อุปกรณ์เครื่องใช้ทุกอย่างจะทำงานได้นั้นจะต้องได้รับการสั่งงานจากมนุษย์อย่างเช่นการเปิด-ปิดวิทยุ การเปิด-ปิดทีวี รวมไปถึงเครื่องจักรกลในงานอุตสาหกรรมต่าง ๆ จะทำงานได้นั้นจะต้องมีมนุษย์คอยควบคุมดูแลและสั่งงาน แต่ปัจจุบันนี้ ความก้าวหน้าทางเทคโนโลยี ทำให้เกิดอุปกรณ์ที่สามารถควบคุมการทำงาน ของอุปกรณ์เครื่องใช้และเครื่องจักรกลต่าง ๆ ให้สามารถทำงานได้เองโดยอัตโนมัติ โดยการทำงานนั้นอาศัยการตัดสินใจจากข้อมูลที่ได้รับมาจากระบบเซนเซอร์ต่าง ๆ ซึ่งปัจจุบันนี้มีเซนเซอร์ที่สามารถตรวจจับสิ่งแวดล้อมต่าง ๆ ได้เกือบทุกชนิด

ระบบไมโครคอนโทรลเลอร์นั้นถูกสร้างขึ้นมาเพื่อให้สามารถควบคุมการทำงานของอุปกรณ์เครื่องมือหรือเครื่องจักรได้แบบอัตโนมัติ โดยไม่ต้องมีคนมาคอยควบคุมดูแลแต่อย่างใด แต่ระบบไมโครคอนโทรลเลอร์สมัยก่อนนั้นมีความยุ่งยากซับซ้อนในการนำมาใช้งานจึงไม่มีการใช้งานที่แพร่หลายมากนัก เมื่อชาวอิตาลีได้นำไมโครคอนโทรลเลอร์มาสร้างเป็นบอร์ดสำเร็จรูป โดยจุดประสงค์คือให้มันสามารถใช้งานได้อย่างง่ายดายเพื่อให้บุคคลทั่วไปนั้น สามารถที่จะนำบอร์ดไมโครคอนโทรลเลอร์สำเร็จรูปนี้มาใช้งานได้ โดยที่ไม่ต้องมีความรู้ความชำนาญในระบบไมโครคอนโทรลเลอร์มากนัก ก็สามารถที่สร้างสรรค์นวัตกรรมใหม่ๆ ขึ้นมาได้

เอกสารคู่มือการใช้งานบอร์ดไมโครคอนโทรลเลอร์สำเร็จรูปที่มีชื่อว่า Arduino นี้ถูกจัดทำขึ้นมาเพื่อให้ผู้ที่ต้องการศึกษาการเขียนโปรแกรมสั่งงานบอร์ดไมโครคอนโทรลเลอร์สำเร็จรูป Arduino ในระดับผู้เริ่มต้นนั้น ได้สามารถเริ่มต้นเรียนรู้ขั้นตอนและวิธีการใช้งานโปรแกรม และเขียนโปรแกรมสั่งงาน Arduino ได้อย่างง่ายดายได้ด้วยตัวเอง ด้วยเนื้อหาที่อ่านเข้าใจได้ง่ายๆ และการเขียนโปรแกรมควบคุมสั่งงานบอร์ด Arduino เพื่อสร้างสรรค์สิ่งต่าง ๆ ด้วยไมโครคอนโทรลเลอร์ จะไม่ใช่เรื่องยากอีกต่อไป

มานพ ปักษี





Arduino อ่านว่าออกเสียงเป็นภาษาอิตาลีว่า “อา-ดู-อิ-โน่” Arduino เป็นบอร์ดสำเร็จรูปที่ทำงานด้วยไอซีไมโครคอนโทรลเลอร์ของ ATMEL AVR ออกแบบโดยชาวอิตาลี เนื่องจากการใช้งานไมโครคอนโทรลเลอร์สมัยก่อนนั้น มีความยุ่งยากต่อการใช้งานมากพอสมควร เพราะนอกจากจะซื้อตัวไอซีไมโครคอนโทรลเลอร์มาแล้ว และ ถึงแม้ว่าจะเขียนโปรแกรมคอมพิวเตอร์เป็น แต่การที่จะนำโปรแกรมที่เขียนขึ้นมานั้น ใส่เข้าไปให้ในตัวไอซี จะต้องแปลงคำสั่งที่เขียนขึ้นมาให้เป็นภาษาของเครื่องคอมพิวเตอร์เสียก่อน เรียกว่าการ คอมไพล์ (Compile) และเมื่อทำการแปลงหรือคอมไพล์แล้วจะได้ไฟล์ที่เป็น .bin หรือ .hex ขึ้นมา

แต่การที่จะนำไฟล์นั้นใส่เข้าไปในตัวไอซีเรียกว่า การเบิร์น หรือ การโปรแกรม (Programmer) จะต้องมีเครื่องโปรแกรม หรือ เครื่องเบิร์น เป็นการลำเลียงไฟล์นั้น ออกจากเครื่องคอมพิวเตอร์ ไปสู่ตัวไอซี โดยเชื่อมต่อผ่านทางพอร์ต Parallel หรือ Serial Port RS232 หรือ USB เพื่อบันทึกข้อมูลนั้นลงในหน่วยความจำของตัวไอซี และจะทำงานตามโปรแกรมที่เราเขียนสั่งงานไว้เมื่อมีไฟเลี้ยงวงจร และราคาของเครื่องมือนั้นก็แพงอยู่ และ เครื่องโปรแกรมแต่ละค่ายก็จะใช้ได้กับไอซีไมโครคอนโทรลเลอร์ของบริษัทนั้นเท่านั้น และถึงแม้จะมีเครื่องโปรแกรมแล้วการที่จะนำไอซีไปใช้งานนั้นก็ต้องมีการออกแบบวงจรและออกแบบ PCB อีก สรุปว่าคนที่จะนำไมโครคอนโทรลเลอร์ไปใช้งานได้นั้นจะต้องมีความรู้ความสามารถในการทำทุกอย่างที่กล่าวมาทั้งหมด

จากการใช้งานที่ยุ่งยากซับซ้อนนี้ ทำให้นาย มาสซิโม บันซี (Massimo Banzi) ชาวอิตาลี และเพื่อนชาวสเปน และ อเมริกาของเขา พัฒนาบอร์ดสำเร็จรูปนี้ขึ้นมา โดยนำไอซีไมโครคอนโทรลเลอร์ ของ ATMEL AVR มาออกแบบวงจรให้พร้อมใช้งานอย่างสำเร็จรูป ทั้งทางฮาร์ดแวร์ และ ซอฟต์แวร์ โดยทางฮาร์ดแวร์นั้น ก็คือการออกแบบวงจรที่สำคัญต่าง เช่นวงจรจ่ายไฟ วงจรผลิตความถี่ และวงจรที่ใช้สำหรับการเชื่อมต่อต่างๆ เพื่อความสะดวกต่อการใช้งาน ส่วนในทางซอฟต์แวร์นั้น ทางทีมงานได้พัฒนาไลบรารี และฟังก์ชันสำเร็จรูปต่างๆมากมายออกมาให้ผู้ใช้งานนั้นเขียนโปรแกรมสั่งงานได้อย่างง่ายดาย ด้วยภาษาที่ใกล้เคียงกับภาษาของมนุษย์ และยังเขียนโปรแกรมบู๊ตโหลดเดอร์ขึ้นมา เพื่อให้สามารถอัพโหลดโปรแกรมเข้าสู่บอร์ดได้โดยไม่ต้องใช้เครื่องโปรแกรมอีกต่อไป โดยการอัพโหลดนั้นจะใช้วิธีการเชื่อมต่อผ่านพอร์ต USB ของคอมพิวเตอร์ทำให้ผู้ใช้งานนั้นสามารถใช้งานได้อย่างสะดวกง่ายดาย และ พวกเขาตั้งชื่อมันว่า Arduino

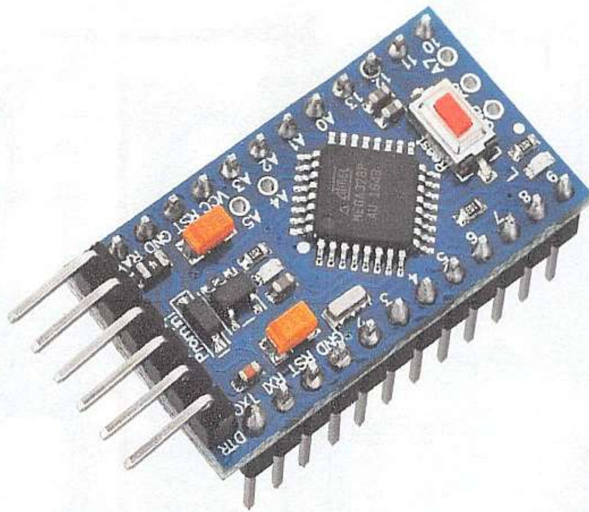


Arduino Team

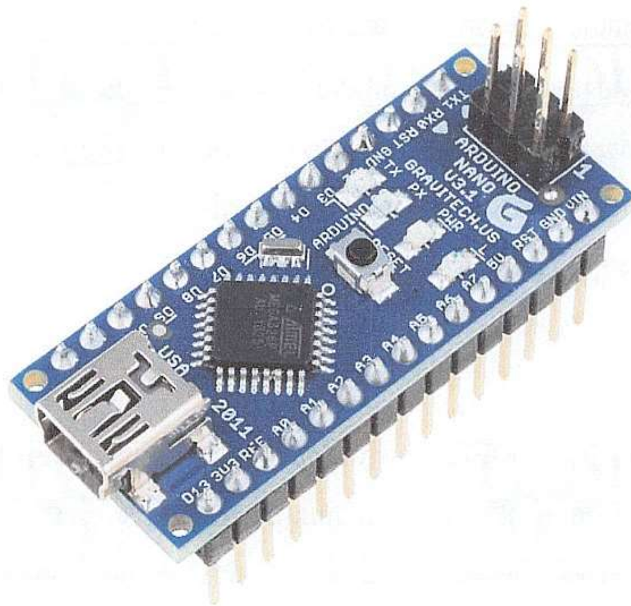
โดยจุดประสงค์หลัก คือ มุ่งหมายให้ใครก็ตาม ที่มีความคิดสร้างสรรค์ ที่อยากสร้างอะไรก็ตามที่ต้องใช้ไมโครคอนโทรลเลอร์ ในการควบคุม สามารถสร้างงานที่เขาคิดออกมาได้ โดยที่บุคคลนั้นไม่ต้องมีความรู้ในด้านนี้มากนัก ไม่ต้องออกแบบวงจรอิเล็กทรอนิกส์เป็น ไม่ต้องออกแบบ PCB เป็น ไม่ต้องมีความรู้เรื่องไมโครคอนโทรลเลอร์ระดับลึก ไม่ต้องรู้จักรีจิสเตอร์ เพียงแค่มีความรู้พื้นฐานอิเล็กทรอนิกส์ในเบื้องต้น และใช้คอมพิวเตอร์เป็น และศึกษาการเขียนโปรแกรมภาษา C เพียงในเบื้องต้นเท่านั้น ก็สามารถใช้บอร์ดสำเร็จรูป Arduino สร้างสิ่งต่าง ๆ ได้อย่างง่ายดาย

Arduino รุ่นต่าง ๆ

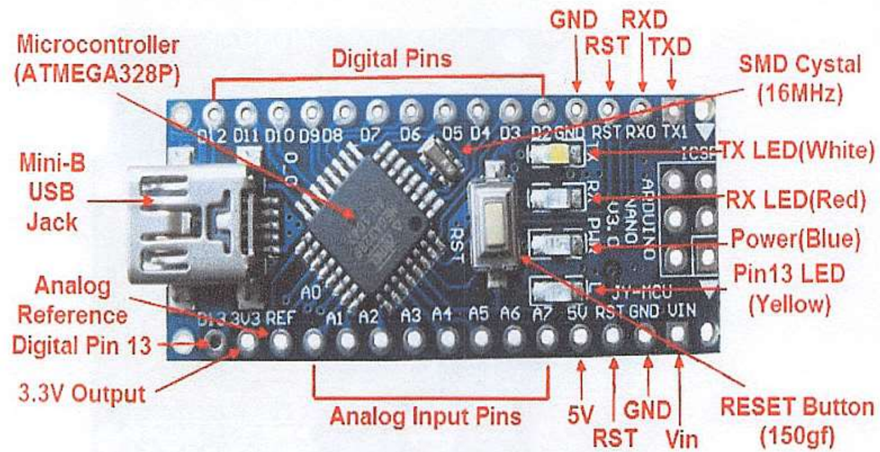
Arduino รุ่นต่าง ๆนั้นมีบอร์ดให้เลือกใช้อยู่มากมายหลายรุ่นมาก การใช้งานก็แล้วแต่งานว่าต้องการความสามารถมากขนาดไหน หากเป็นตัวที่มีหน่วยความจำมาก มีหน่วยประมวลผลที่เร็วแรงกว่า และมีออปชั่นต่าง ๆ มากกว่า ราคาก็จะแพงขึ้นตามไปด้วย เวลาใช้จึงควรเลือกให้พอเหมาะกับงานที่ทำ



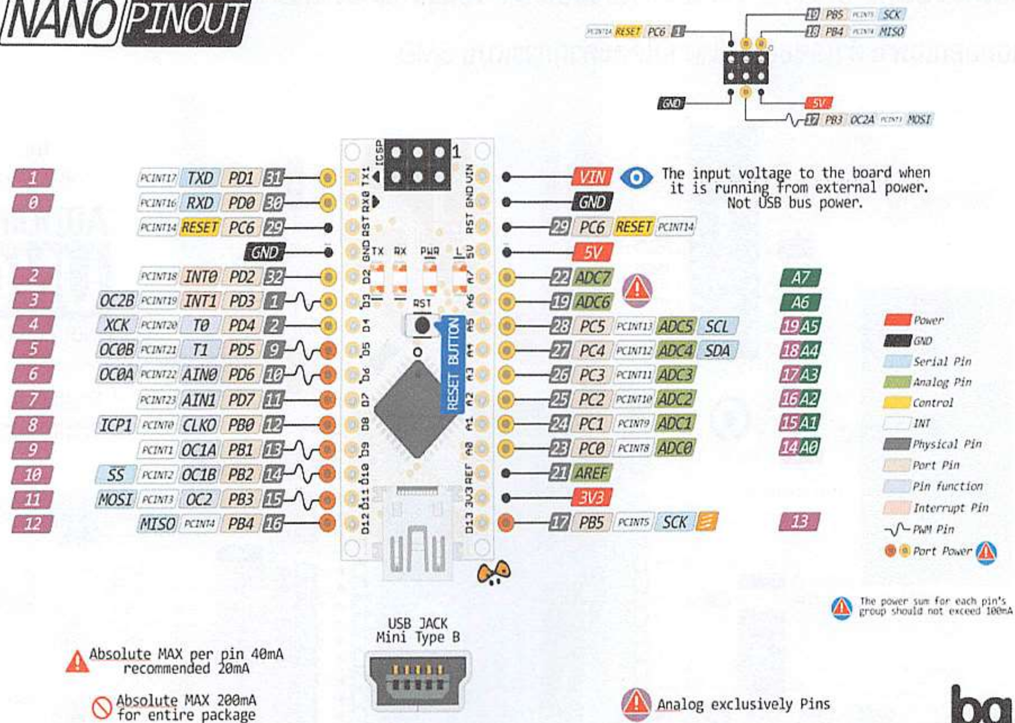
Arduino Pro MINI เป็นรุ่นที่เล็กมาก ใช้ไอซีไมโครคอนโทรลเลอร์เบอร์ ATmega328P แบบ SMD ซึ่งเป็นตัวเดียวกันกับ UNO R3 ดังนั้นคุณสมบัติต่าง ๆ ในการทำงานของทั้งสองบอร์ดนี้จึงเหมือนกันทุกประการ แต่บอร์ดรุ่นนี้ไม่มีไอซีที่ใช้ในการเชื่อมต่อ RS232 ทำให้ไม่สามารถอัพโหลดโปรแกรมผ่านพอร์ต USB ได้ จึงทำให้ไม่สะดวกในการอัพโหลดโปรแกรมลงบอร์ด จึงไม่เหมาะกับมือใหม่ๆ แต่ราคาค่าตัวนั้นอยู่ที่ตัวเลขสองหลักเท่านั้น



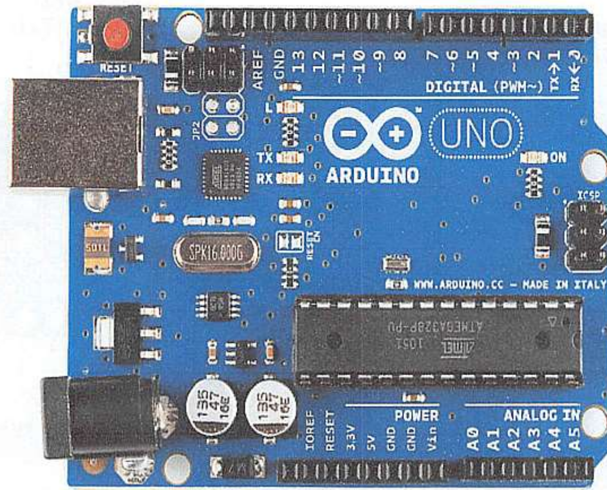
Arduino NANO 3.x เป็นบอร์ดที่ใช้ชิพ Atmaga328P เช่นเดียวกัน เป็นการออกแบบให้เล็กลงมาจาก UNO R3 SMD ดังนั้นคุณสมบัติต่าง ๆ ในการทำงานจึงยังคงเท่าเทียมกันประการ แต่บอร์ดนี้ได้ตัดอุปกรณ์ส่วนที่ไม่จำเป็นออกไปทำให้มีขนาดเล็กลงมาเหลือเพียงน้อยกว่าครึ่งของบอร์ด UNO R3 แต่รุ่นนี้ตอนซื้อต้องดูดี ๆ เพราะบอร์ดรุ่นนี้บางตัวอาจจะใช้ ไอซีไมโครคอนโทรลเลอร์เบอร์ ATmega168 ซึ่งจะมีหน่วยความจำ และหน่วยประมวลผลน้อยกว่าเบอร์ ATmega328 การลดขนาดลงมานั้นทำให้ราคานั้นถูกลงด้วย



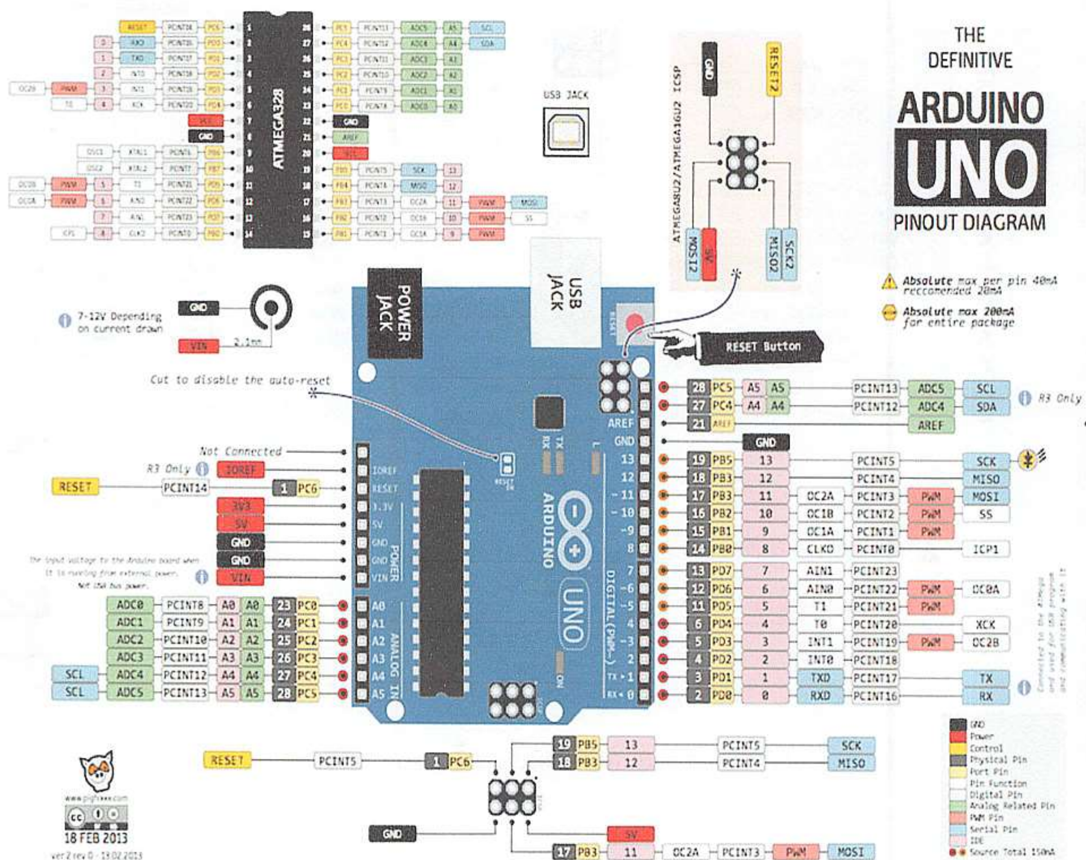
ตำแหน่งขาต่าง ๆ ของบอร์ด NANO V3



ภาพประกอบจากอินเตอร์เน็ต

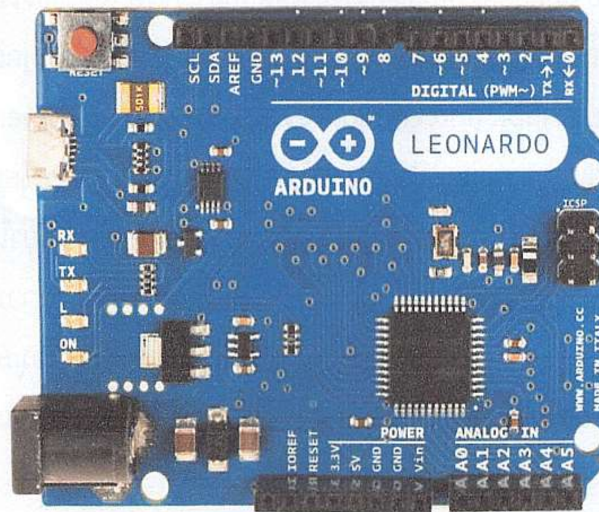


บอร์ด Arduino UNO R3 รุ่นที่เป็นตัวไอซีแบบ DIP เป็นรุ่นที่นิยมใช้กันมากเนื่องจากใช้ไอซีแบบ DIP สามารถถอดออกเฉพาะตัวไอซีออกไปใช้งานได้สะดวกกว่าแบบ SMD

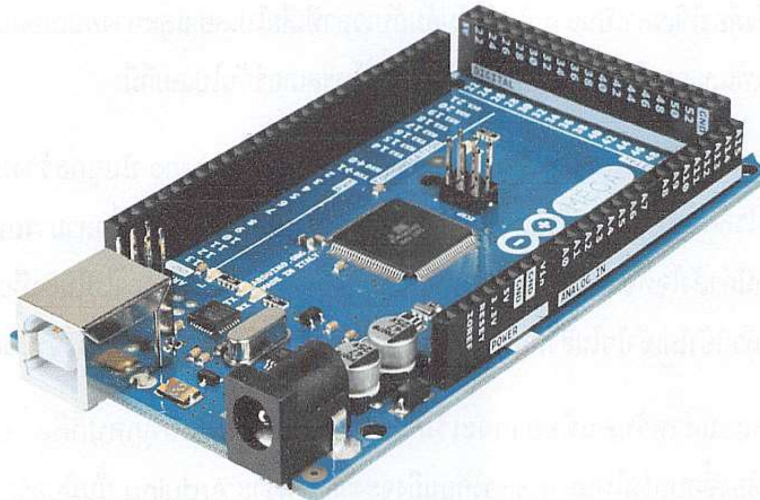


ภาพตำแหน่งรายละเอียดขาต่าง ๆ ในการต่อใช้งานของบอร์ด UNO R3

ภาพประกอบจากอินเตอร์เน็ต



Arduino LEONADO เป็นบอร์ดที่มีขาการใช้งาน และ คุณสมบัติต่าง ๆ นั้นใกล้เคียงกันกับ UNO R3 แถมตำแหน่งขาการใช้งานนั้นเหมือนกันทุกอย่าง ต่างกันที่ใช้ ไอซีไมโครคอนโทรลเลอร์ ATmega32U4 ซึ่งเป็นชิพที่มีโมดูล USB มาให้อยู่แล้วจึงไม่ต้องใช้ชิพแปลงสัญญาณ Serial to USB แต่อย่างใด



Arduino MEGA2560 เป็นรุ่นใหญ่ที่มีคุณสมบัติค่อนข้างสูง มีพอร์ตเชื่อมต่อกับอุปกรณ์อื่นเยอะมากรวมไปถึงออปชั่นพิเศษต่าง ๆ ก็มีมาให้มากขึ้น มีหน่วยความจำมากกว่า และ มีหน่วยประมวลผลที่เร็วกว่า แต่ก็ทำให้มีราคาสูงตามไปด้วย ดังนั้นในการที่จะเลือกบอร์ดใดมาใช้งานนั้น ผู้ใช้ควรเลือกบอร์ดให้เหมาะสมกับงานที่จะทำ เพราะหากเลือกบอร์ดที่มีคุณสมบัติมากมาย และราคานั้นก็จะแพงตามไปด้วย แต่เอามาใช้ทำงานเพียงเล็กน้อยก็ไม่ต่างจากการขี่ช้างจับตั๊กแตน

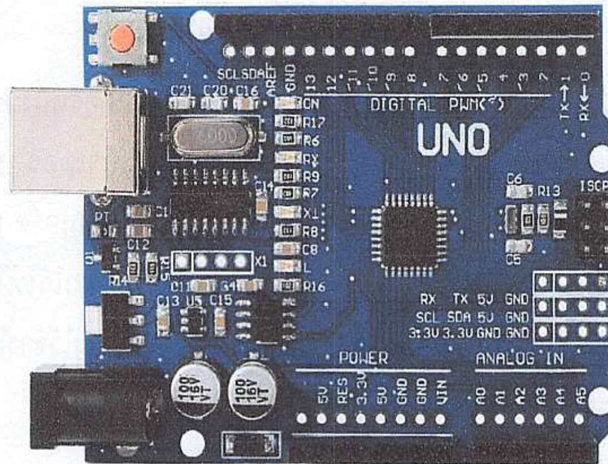
เพราะทุกอย่างถูกออกแบบและสร้างมาให้แบบสำเร็จรูป ไม่ต่างอะไรกับบะหมี่กึ่งสำเร็จรูป ที่เพียง แค่ ฉีกซอง เติมน้ำร้อนและปิดฝารอแค่ 3 นาทีก็อิ่มได้แต่เรื่องรสชาติเราจะไม่พูดถึง แม้กระทั่งเครื่องมือที่ใช้ในการเขียนโปรแกรมที่เรียกว่า IDE (Integrated Development Environment) ก็ถูกสร้างขึ้นมาให้ใช้งานได้แบบสำเร็จรูป และถึงแม้ว่าการเขียนโปรแกรมจะใช้ภาษา C , C++ ในการเขียนสั่งงาน แต่โครงสร้างทั้งหมดถูกสร้างขึ้นมาใหม่แบบสำเร็จรูป ทุก ๆ อย่างมีไลบรารีและตัวอย่างการใช้งานมาให้แบบฟรีๆ แบบว่า ก็อปปี้มาวางก็ใช้งานได้เลยทีเดียว "ย้ำว่ามันง่ายขนาดนั้น" และทุก ๆ อย่างที่กล่าวมานี้เป็นโอเพนซอร์สทั้งหมด ทุกๆอย่างทั้งฮาร์ดแวร์ และซอฟต์แวร์ รวมทั้งวงจรในบอร์ด ลาย PCB ข้อมูลเกี่ยวกับบอร์ด ทุกอย่างเขาเปิดเผยทั้งหมด คำว่า โอเพนซอร์ส หมายถึงใครก็ได้สามารถนำไปใช้ หรือดัดแปลงแก้ไขใดๆก็ได้ โดยไม่มีค่าลิขสิทธิ์ ไม่ต้องเสียเงิน

ถึงแม้ว่าการนำบอร์ด Arduino มาใช้งานนั้นจะง่ายแต่เราก็ไม่ควรละเลยที่จะศึกษาเรียนรู้ถึงพื้นฐานการทำงานของไมโครคอนโทรลเลอร์ อย่างน้อยก็ควรที่จะรู้จักที่มาที่ไปของอุปกรณ์บ้าง เพราะว่าถึงมันจะเป็นบอร์ดสำเร็จรูปที่เขาออกแบบมาให้เราพร้อมใช้งาน แต่หัวใจหลักของมันก็ยังเป็นไอซีไมโครคอนโทรลเลอร์ตัวหนึ่งนั่นเอง ผมเคยเป็นคนหนึ่งที่ไม่ค่อยชอบ บอร์ดสำเร็จรูปแบบนี้เลย เพราะคิดว่ามันจะทำให้เด็ก ๆ สมัยใหม่ขาดความรู้ความเข้าใจการทำงานระดับลึก แต่เมื่อกาลเวลาเปลี่ยนไป เทคโนโลยีเปลี่ยนไป มนุษย์เรามีเวลาน้อยลงไปทุกที หากการที่จะศึกษาวิชาไมโครคอนโทรลเลอร์ เพื่อจะนำมาใช้ทำงานอะไรสักชิ้นหนึ่งต้องใช้เวลาเป็นแรมปี มันไม่คุ้มกับเวลาที่เสียไปเลย และบางคนเมื่อต้องพบเจอกับอะไรที่ยากมาก ๆ ก็ยอมแพ้และถอดใจ จนเลิกศึกษาไมโครคอนโทรลเลอร์กันไปเลยก็มี

หากเราเปิดใจลองมองในมุมที่กลับกันบ้าง เมื่อ Arduino มันถูกสร้างมาให้เราสามารถสร้างสิ่งประดิษฐ์อะไรสักอย่างหนึ่ง แล้วแต่จินตนาการของใครจะคิดออกมาได้ เมื่อสามารถทำได้สำเร็จโดยง่ายดาย มันจะทำให้เรามีกำลังใจที่จะทำสิ่งนั้นต่อไป แล้วเราค่อยๆศึกษามันให้ลึกลงไปในสิ่งที่ยากขึ้นเรื่อย ๆ ถึงแม้ว่าหากศึกษาไปแล้วก็ยังไม่เข้าใจในสิ่งที่ยากกว่านั้น เราก็ยังสามารถใช้งานความสำเร็จรูปของมันได้อยู่ดี

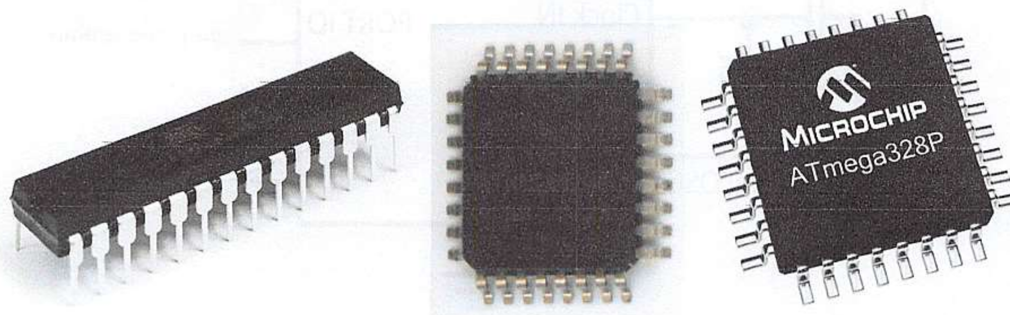
คำถามสำหรับคนที่อยากจะเริ่มศึกษา Arduino เกือบทุกคนก็คือ จะเริ่มต้นยังไง? ต้องรู้อะไรบ้าง? จะต้องซื้อบอร์ดไหนมาทดลองก่อนถึงจะง่าย? เพราะ Arduino นั้นมีบอร์ดให้เลือกใช้งานมากมายหลายรุ่นมาก และราคาก็แตกต่างกัน Arduino นั้นถูกออกแบบและสร้างขึ้นมาโดยชาวอิตาลี ในภาษาอิตาลีนั้น UNO แปลว่า หนึ่ง ดังนั้นบอร์ดที่เขาออกแบบมาสำหรับผู้เริ่มต้นมันจึงเป็นบอร์ดรุ่น UNO นั่นเอง

และบอร์ดที่ผู้เขียนเลือกมาใช้เป็นตัวอย่างในหนังสือเล่มนี้คือ UNO R3 SMD เนื่องจากเป็นรุ่นที่มีราคาถูกมาก เหมาะแก่การนำมาเรียนรู้ เพราะหากผู้ที่ต้องการศึกษานั้นเกิดยอมแพ้ถอดใจล้มเลิกไป ก็จะไม่เป็นภาระในเรื่องค่าใช้จ่ายที่มากเกินไป และถึงแม้บอร์ดนี้จะใช้เป็นชิพแบบบัดกรีติดแผ่นวงจร แต่สำหรับช่างอิเล็กทรอนิกส์แล้ว การถอดเปลี่ยนตัวชิพนั้นก็ไม่ใช่เรื่องยาก



หน้าตาบอร์ด Arduino รุ่น UNO R3 SMD

บอร์ด UNO R3 SMD เป็นบอร์ดรุ่นที่พัฒนาเพื่อลดต้นทุนลง มาโดยการใช้ตัวไอซีแบบชิพ SMD โดยไอซีที่เป็นไมโครคอนโทรลเลอร์ ที่เป็นหัวใจหลักก็ยังคงใช้เบอร์ **ATMEGA328P** แต่เดิมเป็นของบริษัท ATMEL แต่ปัจจุบันนี้ได้กลายเป็นของบริษัทไมโครชิพไปเรียบร้อยแล้ว



MICROCHIP

English | Search

PRODUCTS | APPLICATIONS | DESIGN SUPPORT | SAMPLE AND BUY | ABOUT US | CONTACT US | MYMICROCHIP LOGIN

ATmega328P DATA SHEET (12/10/2016) DOWNLOAD

BUY IT NOW

ATmega328P

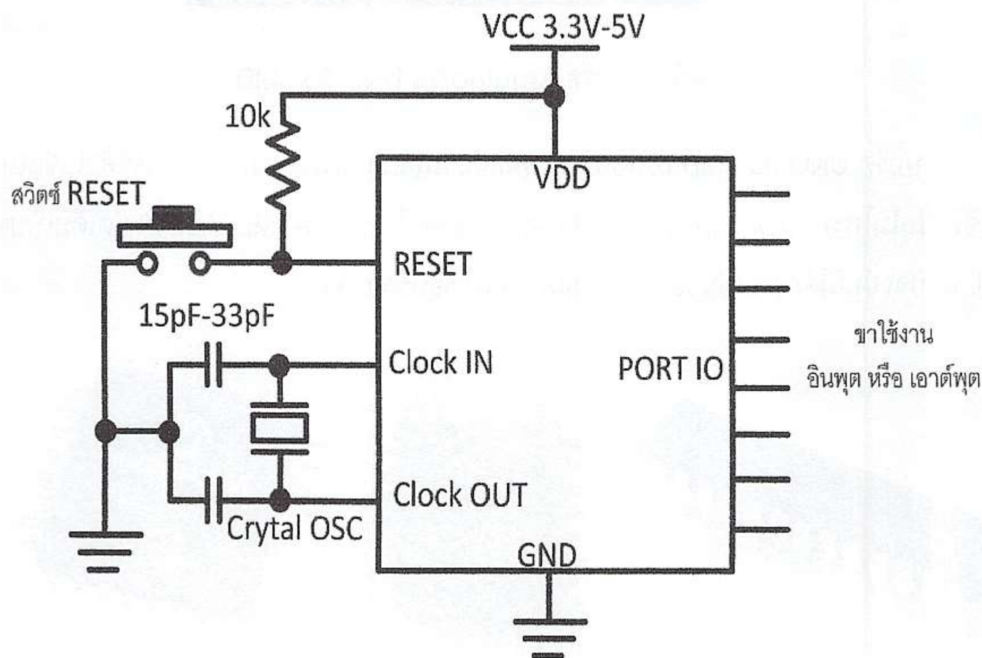
In Production

The high-performance Microchip picoPower 8-bit AVR RISC-based microcontroller combines 32KB ISP flash memory with read-while-write capabilities, 1024B EEPROM, 2KB SRAM, 23 general purpose I/O lines, 32 general purpose working registers, three flexible timer/counters with compare modes, internal and external interrupts, serial programmable USART, a byte-oriented 2-wire serial interface, SPI serial port, a 6-channel 10-bit A/D converter (8-channels in TQFP and QFN/MLF packages), programmable watchdog timer with internal oscillator, and five software selectable power saving modes. The device operates between 1.8-5.5 volts.

By executing powerful instructions in a single clock cycle, the device achieves throughputs approaching 1 MIPS per MHz, balancing power consumption and processing speed.

ไมโครคอนโทรลเลอร์คืออะไร

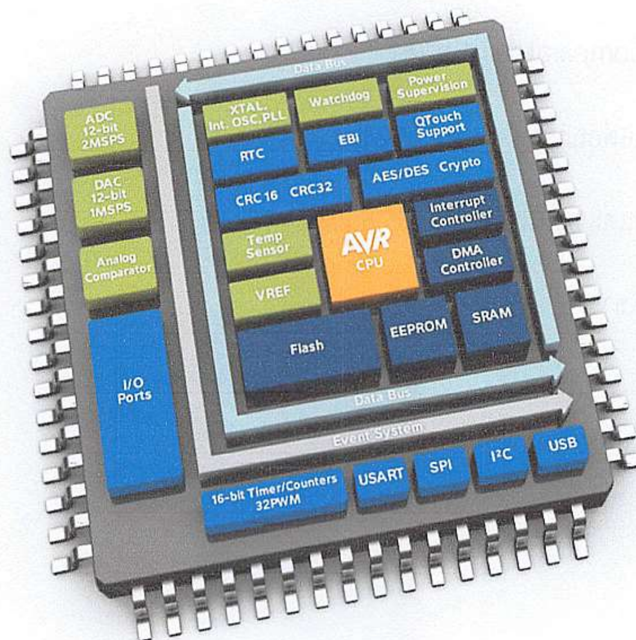
ไอซีไมโครคอนโทรลเลอร์นั้นเปรียบเทียบได้กับคอมพิวเตอร์หนึ่งเครื่องเลยทีเดียว เพราะภายในไอซีไมโครคอนโทรลเลอร์นั้น ได้บรรจุทุก ๆ อย่างที่มีอยู่ในคอมพิวเตอร์ เช่น หน่วยประมวลผล CPU หน่วยความจำทั้งแบบ ROM และ RAM รวมทั้ง EEPROM และ ยังมีพอร์ต PORT อินพุต-เอาต์พุตเพื่อใช้เชื่อมต่อกับอุปกรณ์อื่น ๆ ได้ และภายในนั้นได้รวมวงจรต่าง ๆ เอาไว้ให้ใช้งานพร้อมเสร็จ ทั้ง ADC, Comparator ,TIMER ,PWM ,I2C ,USART ,SPI ทุกอย่างถูกยัดรวมไว้ข้างในเอาไว้เสร็จสรรพ การใช้งานก็เพียงแค่เขียนโปรแกรมสั่งงานเท่านั้น



แต่ก่อนที่ไมโครคอนโทรลเลอร์จะสามารถทำงานได้นั้นต้องมีส่วนประกอบหลักๆอยู่ 5 ข้อ

1. **ไฟเลี้ยงวงจร** ไมโครคอนโทรลเลอร์ก็เป็นไอซีตัวหนึ่ง การที่ไอซีจะทำงานได้ต้องมีไฟเลี้ยงวงจรเช่นเดียวกับตัวไอซีทั่วไป ไมโครคอนโทรลเลอร์ส่วนใหญ่จะใช้ไฟเลี้ยงประมาณ 5V และบางตัวอาจใช้ 3.3V และรุ่นใหม่ๆบางตัวใช้ไฟเลี้ยงแค่ 1.8V เท่านั้น
2. **หน่วยผลิตความถี่สัญญาณนาฬิกา** หรือ Clock ไมโครคอนโทรลเลอร์จะต้องมีสัญญาณนาฬิกาเพื่อใช้อ้างอิงในการทำงานและประมวลผลต่าง ๆ หากไม่มีสัญญาณนาฬิกาแล้ว ไมโครคอนโทรลเลอร์ ก็จะไม่สามารถทำงานได้ ส่วนใหญ่จะใช้คริสตอลเป็นตัวผลิตความถี่ ไมโครคอนโทรลเลอร์ส่วนใหญ่ จะทำงานที่ความถี่ตั้งแต่ 4MHz - 20MHz และไมโครคอนโทรลเลอร์บางตัว อาจจะมีวงจรผลิตความถี่ภายในของตัวเองโดยไม่ต้องอาศัยวงจรผลิตความถี่จากภายนอกเลยก็ได้ แต่ ในบอร์ด Arduino UNO R3 นั้นใช้คริสตอลในการผลิตความถี่ 16MHz

3. **วงจรรีเซ็ต Reset** วงจรรีเซ็ตนั้นเป็นวงจรเพื่อสั่งให้ไมโครคอนโทรลเลอร์ เริ่มต้นทำงานตามคำสั่งโปรแกรมตั้งแต่แรก และไม่ว่าโปรแกรมนั้นจะทำงานอะไรอยู่ก็ตาม หากมีการกดปุ่ม Reset ไมโครคอนโทรลเลอร์จะกลับไปเริ่มต้นทำงานใหม่ตั้งแต่แรก และ หากไม่มีวงจรรีเซ็ต โมโครคอนโทรลเลอร์จะทำงานไม่ได้
4. **DATA หรือ ข้อมูล** ข้อมูลมีอยู่ด้วยกันสองชนิด ชนิดแรกเป็นข้อมูลหลัก ที่เป็นเฟิร์มแวร์คือโปรแกรมหลักที่เราเขียนสั่งงานเอาไว้ให้ทำงานต่าง ๆ ข้อมูลนี้จะถูกเก็บเข้าไว้ในหน่วยความจำหลักของไมโครคอนโทรลเลอร์ในขั้นตอนการเบิร์นโปรแกรม ข้อมูลนี้จะถูกเก็บอยู่ในหน่วยความจำชนิด ROM ข้อมูลชนิดนี้จะยังคงอยู่ถึงแม้ว่าจะไม่มีไฟเลี้ยงวงจรก็ตาม ส่วนข้อมูลอีกชนิดหนึ่งนั้นเป็นข้อมูลที่ใช้ในการประมวลผลเป็นข้อมูลชั่วคราว ข้อมูลส่วนนี้ได้มากจากการรับค่าจากเซนเซอร์ต่าง ๆ ที่รับเข้ามาทางขาอินพุต หรือค่าตัวแปรที่ถูกสร้างขึ้นในโปรแกรมเพื่อการคิดคำนวณ และการประมวลผลต่าง ๆ ข้อมูลส่วนนี้จะถูกเก็บอยู่ในหน่วยความจำแบบ RAM เป็นการเก็บข้อมูลแบบชั่วคราว และข้อมูลเหล่านี้จะหายไปเมื่อไม่มีไฟเลี้ยงวงจร
5. **PORT IO** คือพอร์ตอินพุตและเอาต์พุต โดย I หมายถึง INPUT และ O หมายถึง OUTPUT ดังนั้น PORT IO ก็คือ พอร์ตอินพุต และ เอาต์พุต ของไมโครคอนโทรลเลอร์ พอร์ต หรือ ขาอินพุต ของไมโครคอนโทรลเลอร์ที่มีไว้สำหรับต่อกับระบบเซนเซอร์ต่าง ๆ เพื่อรับข้อมูลเข้ามาเพื่อประมวลผล และ ใช้พอร์ตเอาต์พุตส่งข้อมูลออกไปเพื่อควบคุมอุปกรณ์ต่าง ๆ ให้ทำงานตามวัตถุประสงค์



ไมโครคอนโทรลเลอร์ที่ใช้ในบอร์ด UNO R3 SMD

ไอซีไมโครคอนโทรลเลอร์ที่ใช้ในบอร์ด UNO R3 SMD นั้นเป็นไอซีเบอร์ ATMEGA328P เป็นไมโครคอนโทรลเลอร์ขนาด 8 บิต ใช้หน่วยความจำในการเก็บโปรแกรมหลักเป็นแบบแฟลช (Flash)

มีหน่วยความจำที่ใช้เก็บโปรแกรมหลัก 32 KB

มีหน่วยประมวลผล CPU ที่มีความเร็ว 20 MIPS

มีหน่วยความจำชั่วคราว RAM 2,048 BYTE หรือ 2 KB

มีหน่วยความจำเก็บข้อมูล EEPROM 1024 BYTE หรือ 1 KB

มีพอร์ตที่ใช้ในการติดต่อส่อสารข้อมูลอนุกรม แบบ UART 1 ช่อง

มีพอร์ตที่ใช้ในการติดต่อส่อสารข้อมูลอนุกรม แบบ SPI 2 ช่อง

มีพอร์ตที่ใช้ในการติดต่อส่อสารข้อมูลอนุกรม แบบ I2C 1 ช่อง

มีพอร์ตสำหรับจับสัญญาณ PWM อินพุต 1 ช่อง

มีพอร์ตสำหรับส่งสัญญาณ PWM เอาต์พุต 6 ช่อง

มีไทม์เมอร์ (TIMER) ขนาด 8 บิต 2 ตัว

มีไทม์เมอร์ (TIMER) ขนาด 16 บิต 1 ตัว

มีคอมพาราเตอร์ (Comparators) 1 ช่อง

สามารถทำงานได้ตั้งแต่อุณหภูมิ -40 ถึง 85 องศาเซลเซียส

สามารถทำงานได้ตั้งแต่แรงดันไฟเลี้ยง 1.8-5.5 โวลต์

และมีขาทั้งหมด 32 ขา

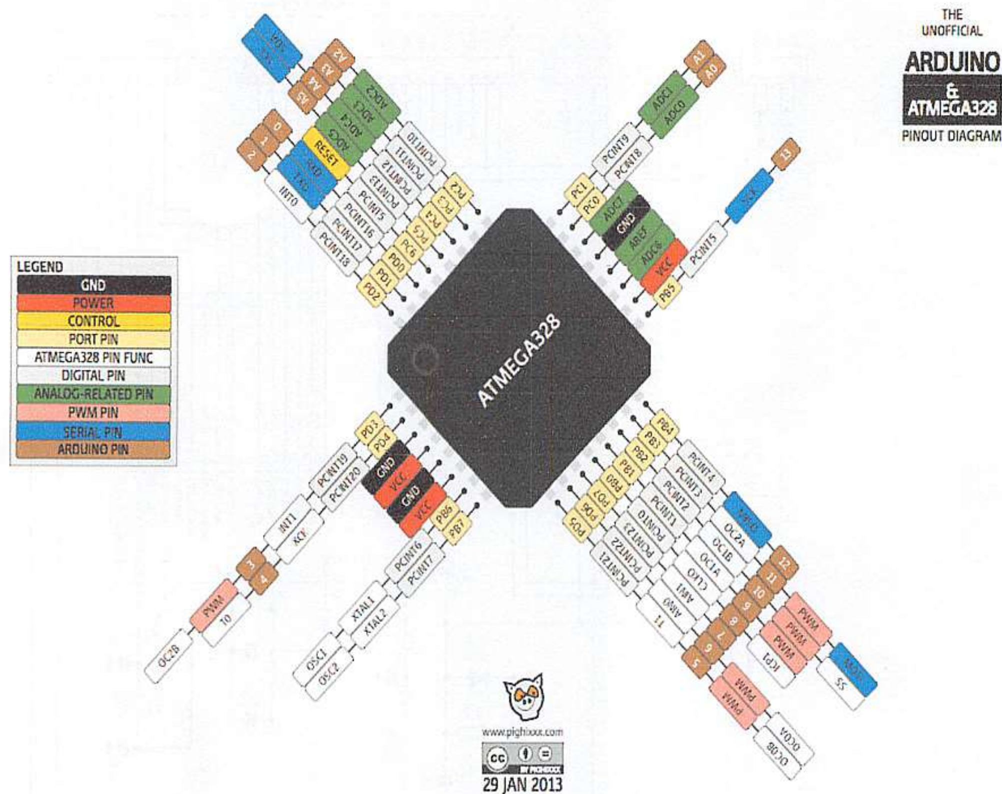
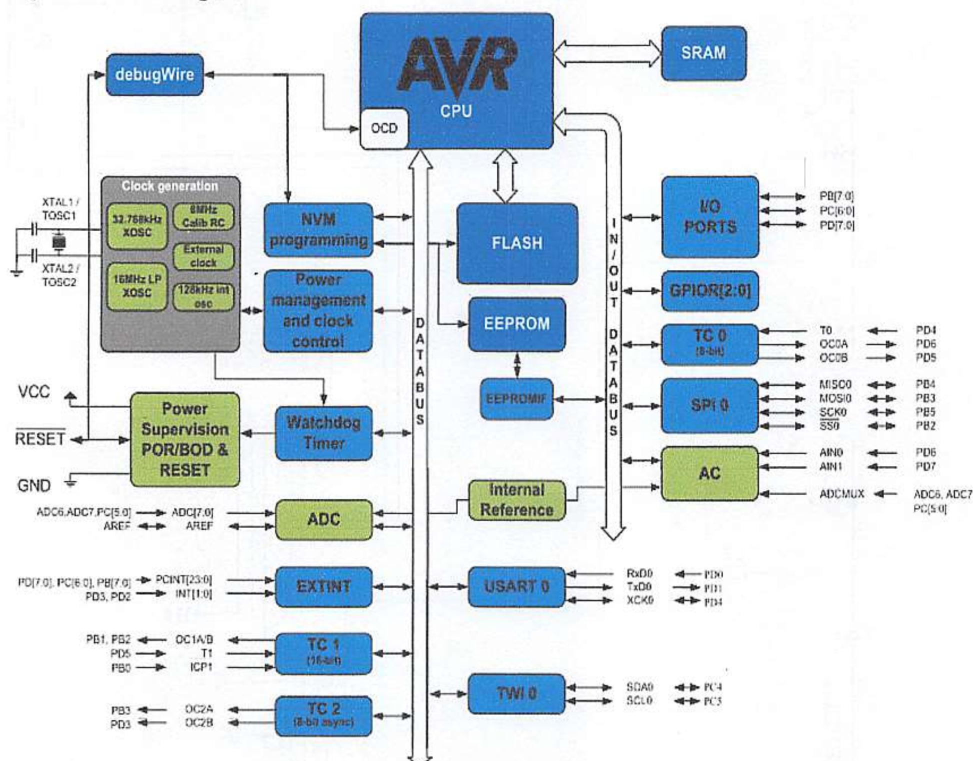
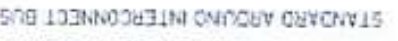


Figure 4-1. Block Diagram



บล็อกไดอะแกรมของไอซีไมโครคอนโทรลเลอร์ ATMEGA328P


STANDARD ARDUINO INTERCONNECT BUS

ช่องต่อไฟเลี้ยง 7-12V

USB

สวิตซ์ RESET

ขาที่ใช้รับ-ส่งข้อมูล
แบบดิจิตอล
มี 14 ขา คือ 0-13

ขารับสัญญาณอินพุต
แบบอนาล็อก A0-A5
มีทั้งหมด 6 ช่อง

เอาไว้ต่อสายโปรแกรมแบบ ICSP

ภาพแสดงจุดต่อใช้งานต่าง ๆ บนบอร์ด UNO R3 SMD

ไฟออก 5V	5V
ขา RESET	RES
ไฟออก 3.3V	3.3V
ไฟออก 5V	5V
GND = กราวด์	GND
ไฟเข้า 7-12V	VIN

ADC 10 bit

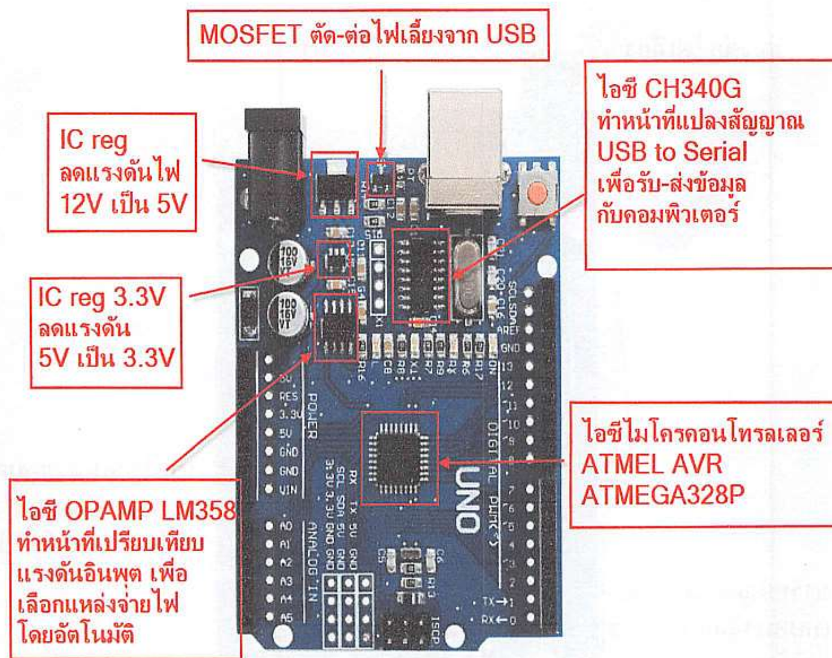
รับสัญญาณ
แบบอนาล็อก

A0
A1
A2
A3
A4
A5

SCL
SDA ← I2C
รับ-ส่งข้อมูล

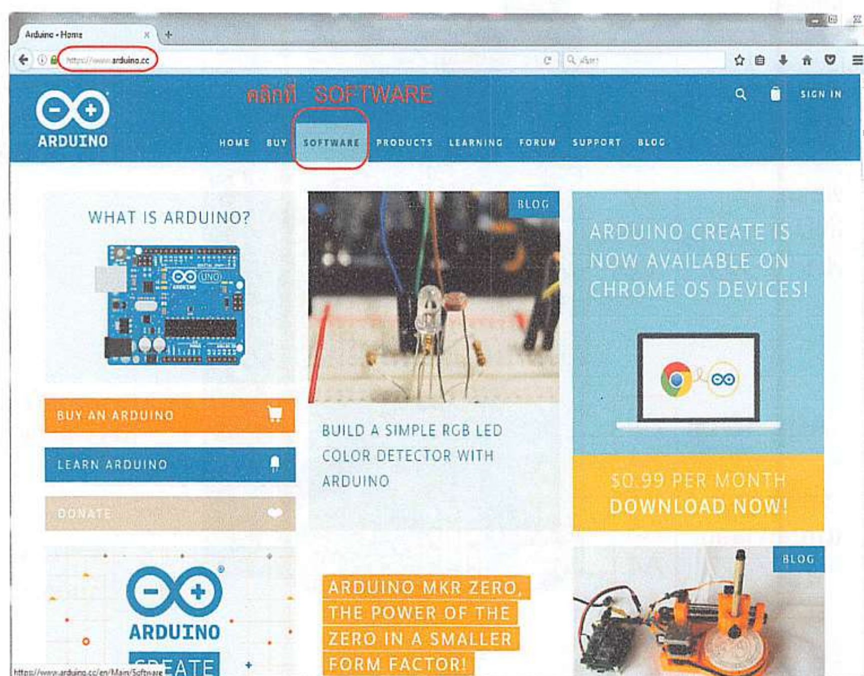
11
10
9 ← PWM
6 สร้างสัญญาณพัลส์
5
3

TX ← UART
RX รับ-ส่งข้อมูล

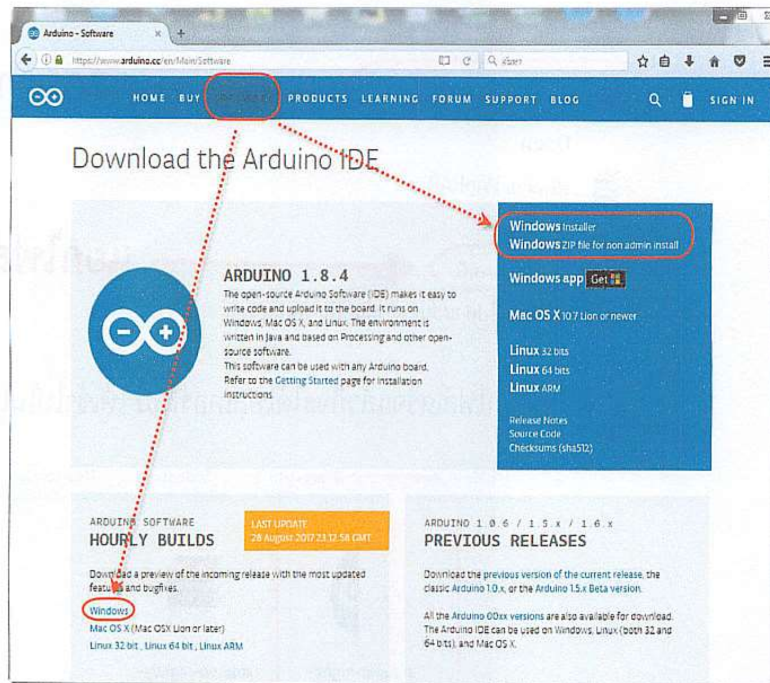


การดาวน์โหลดโปรแกรม

สำหรับโปรแกรมที่ใช้ในการเขียนโค้ดโปรแกรมเพื่อสั่งงานบอร์ด Arduino นั้นให้เข้าไปที่เว็บไซต์หลักของ Arduino ที่ https://www.arduino.cc



เมื่อเข้าไปแล้วให้คลิกที่เมนู SOFTWARE ด้านบน เมื่อเข้ามาหน้าดาวน์โหลดแล้วให้เลื่อนสกอร์บาร์ด้านขวามือลงมานิดหน่อยก็จะพบไฟล์ที่เราต้องการจะดาวน์โหลด



ถึงแม้ว่าการใช้ โปรแกรม Arduino IDE นั้นเราจะเขียนสั่งงานไมโครคอนโทรลเลอร์ด้วยภาษา C, C++ แต่ตัวโปรแกรม Arduino IDE ที่เราใช้เขียนนั้นมันถูกเขียนขึ้นมาด้วยภาษา JAVA ดังนั้นโปรแกรมจึงไม่เลือกระบบปฏิบัติการ ที่ใช้ ไม่ว่าจะเป็น Windows Linux หรือ Mac สรุปว่า ไม่ว่าคอมพิวเตอร์จะเป็นเครื่องที่ใช้งานระบบไหนก็สามารถนำมาเขียนโปรแกรม Arduino ได้ทั้งนั้น

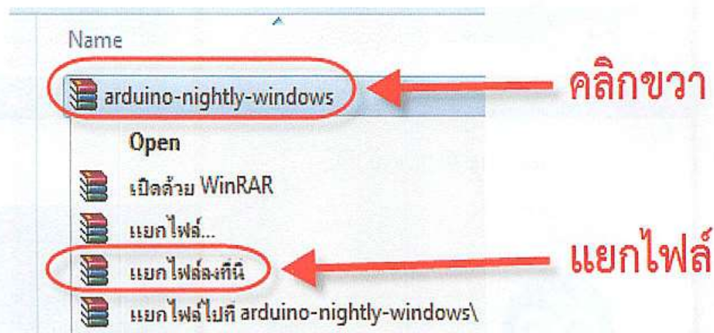
เมื่อตัวโปรแกรมสามารถรันได้ทุกระบบปฏิบัติการแต่การดาวน์โหลดนั้นก็ต้องเลือกไฟล์ให้ตรงกับระบบปฏิบัติการที่ใช้ด้วย เช่นคนที่ใช้ระบบ Windows ก็ให้เลือกดาวโหลดไฟล์สำหรับ Windows คนที่ใช้เครื่อง Mac ก็ให้ดาวโหลดไฟล์ที่ใช้สำหรับเครื่อง Mac และส่วนใครที่ใช้ระบบปฏิบัติการ Linux ก็ให้โหลดไฟล์ที่ใช้สำหรับระบบ Linux

ในส่วนของไฟล์ที่ใช้สำหรับ Windows นั้นมีสองแบบให้เลือกใช้

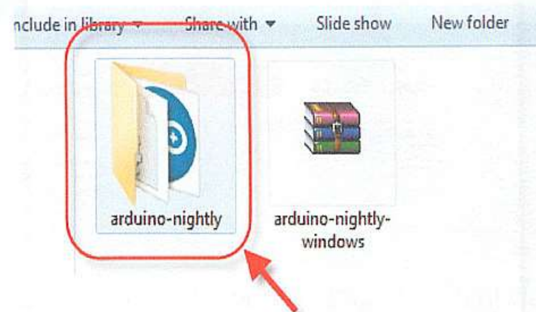
1.เป็นไฟล์ Windows Installer เป็นไฟล์ที่โหลดมาแล้วต้องติดตั้งเหมือนกับโปรแกรมทั่วๆไป

2.ส่วนเป็นไฟล์ที่บีบอัดเป็น Zip ให้คลิกที่ คำว่า Windows หรือ Windows zip file for non admin install ตามรูปข้างบน

ไฟล์ที่เป็น zip นั้นเมื่อเราแยกไฟล์ออกมาแล้วก็ใช้งานได้ทันที โดยเราสามารถแยกไฟล์ไว้ที่ไดร์หรือโฟลเดอร์ใดก็ได้ โดยการแยกไฟล์นั้นหากมีโปรแกรมแยกไฟล์ winrar ภาษาไทย ก็คลิกขวาที่ไฟล์ที่โหลดมาแล้วเลือก แยกไฟล์ลงที่นี่ หรือภาษาอังกฤษ ก็จะเป็น Extract Here



เมื่อทำการแยกไฟล์เสร็จแล้วก็จะได้โฟลเดอร์ใหม่ ให้เข้าไปในโฟลเดอร์

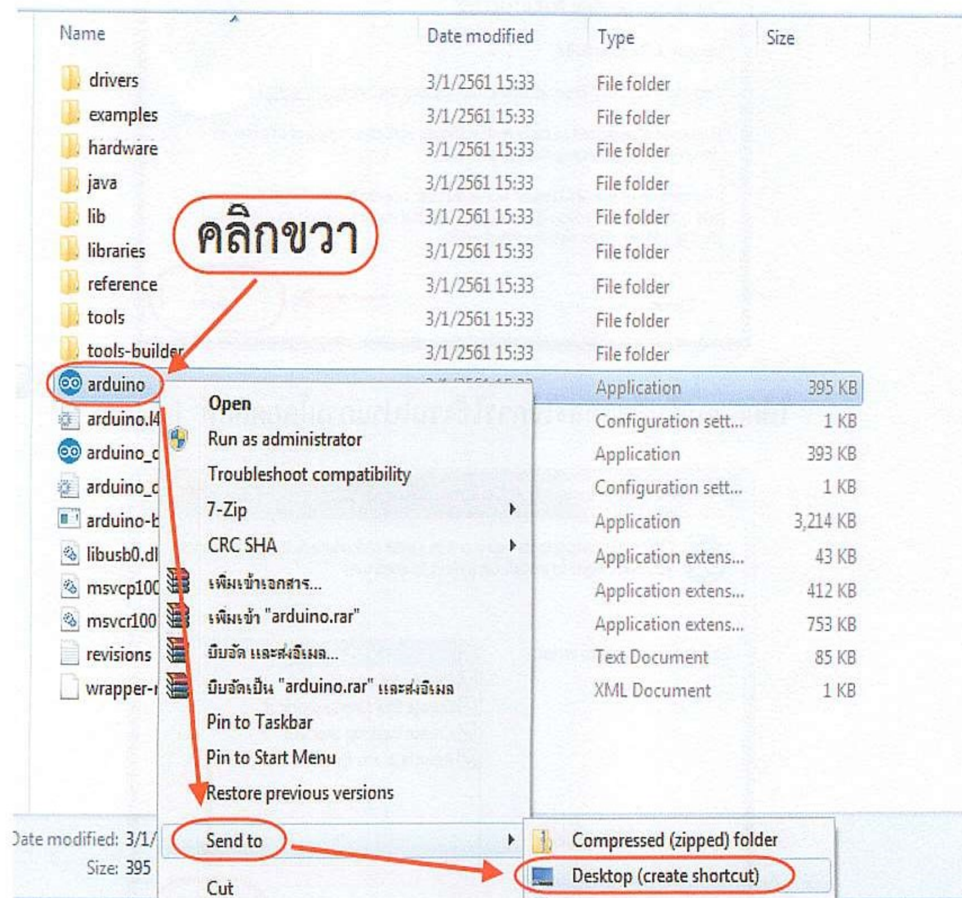


เมื่อแยกไฟล์เสร็จจะได้โฟลเดอร์ใหม่

เมื่อเข้าดูในโฟลเดอร์ก็จะพบไฟล์มากมาย แต่ตัวที่จะเรียกใช้งานโปรแกรมนั้นมีชื่อว่า Arduino และสามารถเปิดใช้งานได้ทันที

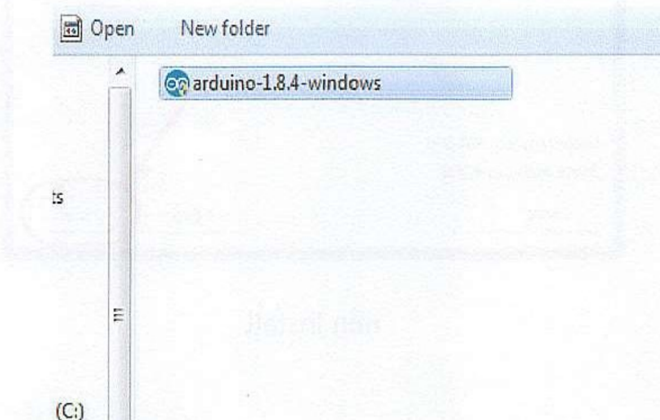
Name	Date modified	Type	Size
drivers	3/1/2561 15:33	File folder	
examples	3/1/2561 15:33	File folder	
hardware	3/1/2561 15:33	File folder	
java	3/1/2561 15:33	File folder	
lib	3/1/2561 15:33	File folder	
libraries	3/1/2561 15:33	File folder	
reference	3/1/2561 15:33	File folder	
tools	3/1/2561 15:33	File folder	
tools-builder	3/1/2561 15:33	File folder	
arduino	3/1/2561 15:33	Application	395 KB
arduino.l4j	3/1/2561 15:33	Configuration sett...	1 KB
arduino_debug	3/1/2561 15:33	Application	393 KB
arduino_debug.l4j	3/1/2561 15:33	Configuration sett...	1 KB
arduino-builder	3/1/2561 15:33	Application	3,214 KB
libusb0.dll	3/1/2561 15:33	Application extens...	43 KB
msvcp100.dll	3/1/2561 15:33	Application extens...	412 KB
msvcr100.dll	3/1/2561 15:33	Application extens...	753 KB
revisions	3/1/2561 15:33	Text Document	85 KB
wrapper-manifest	3/1/2561 15:33	XML Document	1 KB

แต่ถ้าหากต้องการที่จะสร้างทางลัดไว้ที่หน้าจอเพื่อความสะดวกในการเรียกใช้งานก็ให้ทำการคลิกขวา แล้วเลือกไปที่ Send to → Desktop ก็จะได้ไอค่อนที่หน้าจอให้เรียกใช้งานได้สะดวกขึ้น



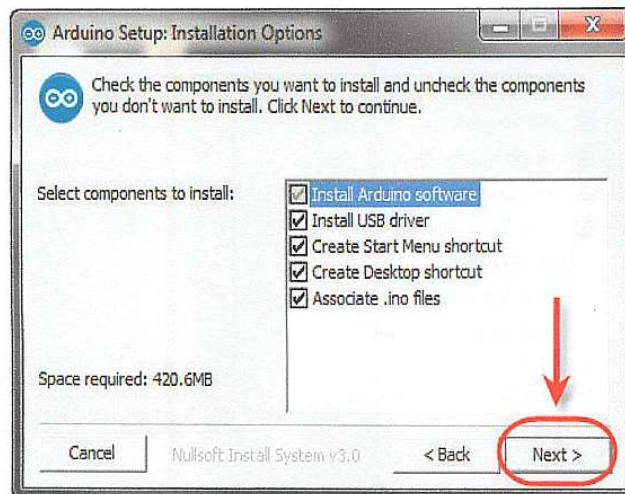
ส่วนไฟล์ที่เป็น แบบ Windows Install จะต้องติดตั้งตามวิธีการเหมือนโปรแกรมทั่วไป แนะนำว่าหากไม่ค่อยมีความรู้เรื่องคอมพิวเตอร์มากนักก็ให้ดาวน์โหลดไฟล์ที่เป็น install จะไม่ค่อยมีปัญหา เพราะโปรแกรมจะทำการติดตั้งสิ่งที่จำเป็น และโฟลเดอร์ที่เก็บไฟล์มาตรฐานต่างๆให้ ในขั้นตอนของการติดตั้ง โดยให้ทำการติดตั้งตามวิธีการต่อไปนี้

ให้ดับเบิ้ลคลิกไปที่ไฟล์ตัวติดตั้งที่โหลดมาดังรูป

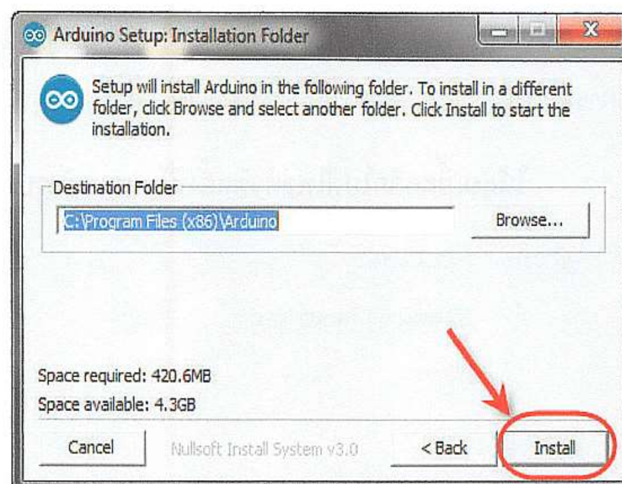




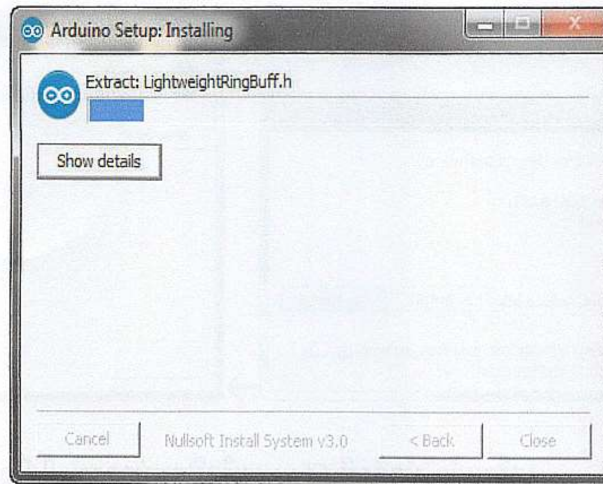
ให้กดยอมรับข้อตกลงในการใช้งานโปรแกรมโดยคลิกที่ I Agree



คลิก Next

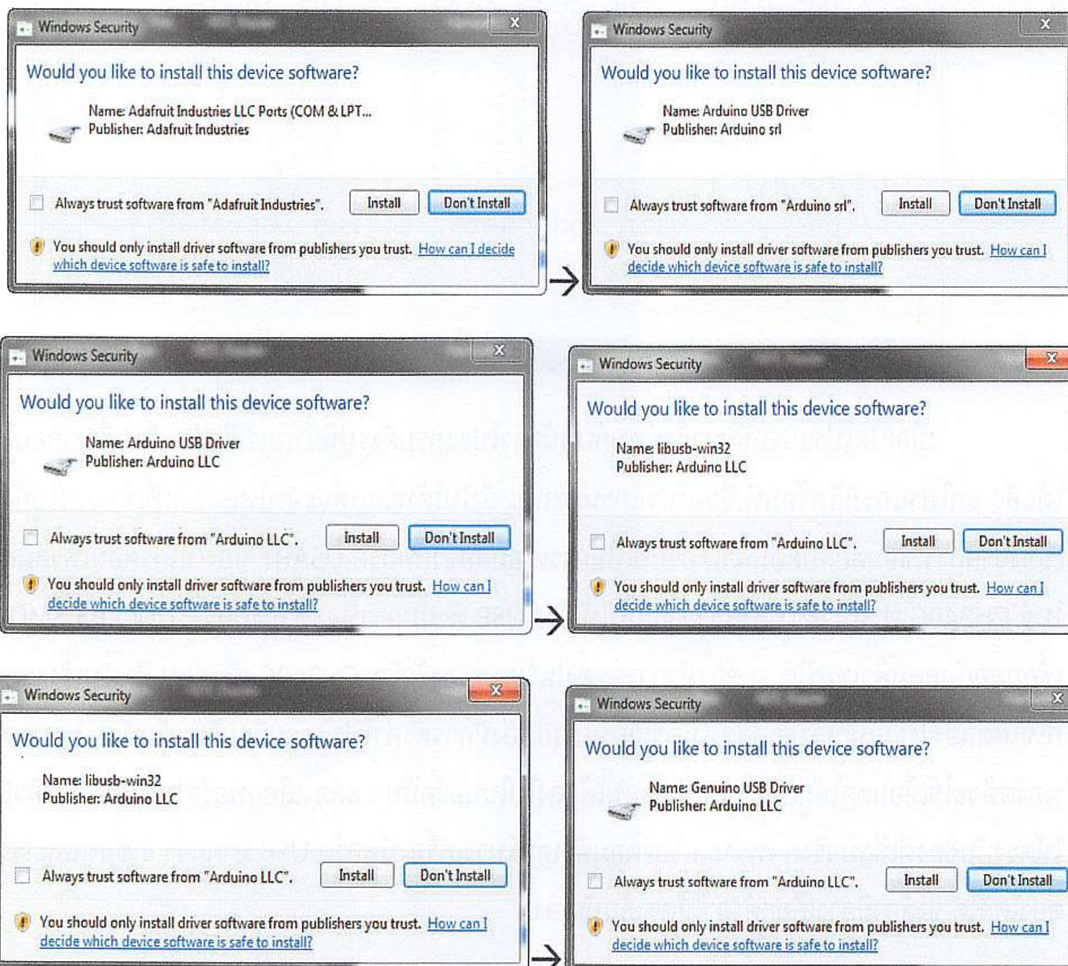


คลิก Install



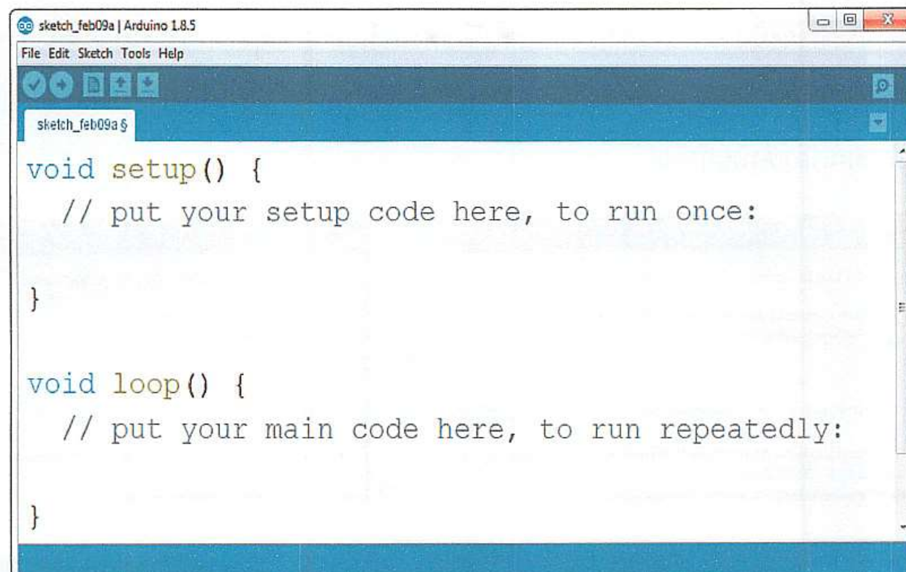
นั่งรอครับ

ระหว่างนี้จะมีหน้าต่างเด้งขึ้นมาให้ติดตั้งไดร์เวอร์ต่างๆของบอร์ด หากต้องการติดตั้งก็ให้คลิก Install ไปเรื่อยๆครับ หรือไม่อยากติดตั้งก็กด Don't Install ครับ เพราะไดร์เวอร์ของบอร์ดนั้นเราสามารถที่จะดาวน์โหลดมาติดตั้งทีหลังก็ได้





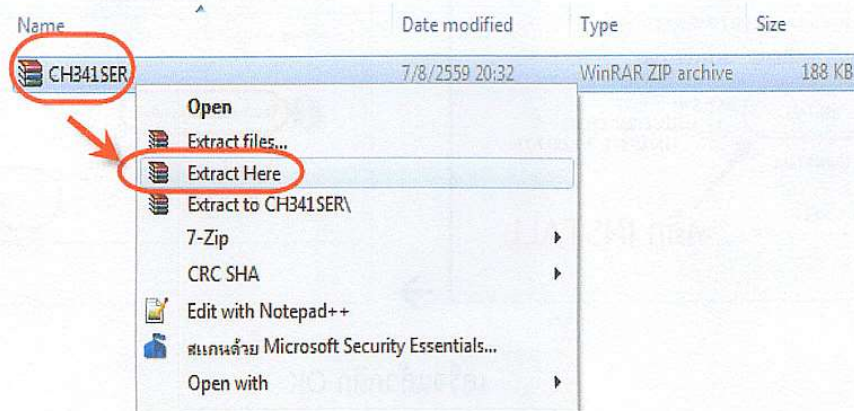
เสร็จแล้วให้คลิกที่ Close เพื่อปิดหน้าต่างไปได้เลย



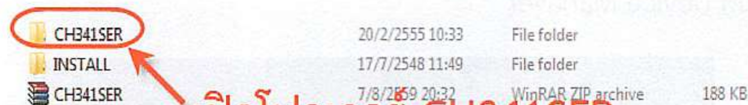
เมื่อเปิดโปรแกรมขึ้นมาก็จะพบกับหน้าตาโปรแกรมดังรูปที่ผ่านมา ถือว่าพร้อมที่จะเขียนโปรแกรมได้แล้ว แต่โปรแกรมก็พร้อมที่เขียนโปรแกรมเท่านั้น ยังไม่ได้พร้อมที่จะอัพโหลดโค้ดที่เขียนลงไปยังบอร์ดได้ เนื่องจากการเชื่อมต่อกับคอมพิวเตอร์นั้นเป็นการเชื่อมต่อจากพอร์ต USART ของไมโครคอนโทรลเลอร์แล้วใช้ไอซี CH340G ทำหน้าที่แปลงสัญญาณนั้นให้เป็น USB ดังนั้นการที่จะทำให้บอร์ด UNO R3 SMD สามารถเชื่อมต่อกับคอมพิวเตอร์ได้ จะต้องทำการติดตั้งไดร์เวอร์ของตัวชิพ CH340G เสียก่อน ติดตั้งครั้งแรกครั้งเดียวก็ใช้ได้ตลอดไป และหลังจากติดตั้งไดร์เวอร์แล้วก็ต้องมีการตั้งค่าโปรแกรมอีกเพียงเล็กน้อย เพราะต้องเลือกพอร์ตที่จะใช้เชื่อมต่อให้ถูกต้อง เพื่อที่จะอัพโค้ดลงไปในบอร์ดนั้น จะต้องเลือกพอร์ตในการเชื่อมที่ตัวโปรแกรมให้ตรงกับช่องที่เสียบจริงๆ เพราะเครื่องคอมพิวเตอร์บางเครื่องมีพอร์ต USB มากกว่า 4 ช่อง และเราจะเสียบช่องใดก็ได้ จึงต้องมีการเลือกตั้งค่าให้ตรงกันนั่นเอง

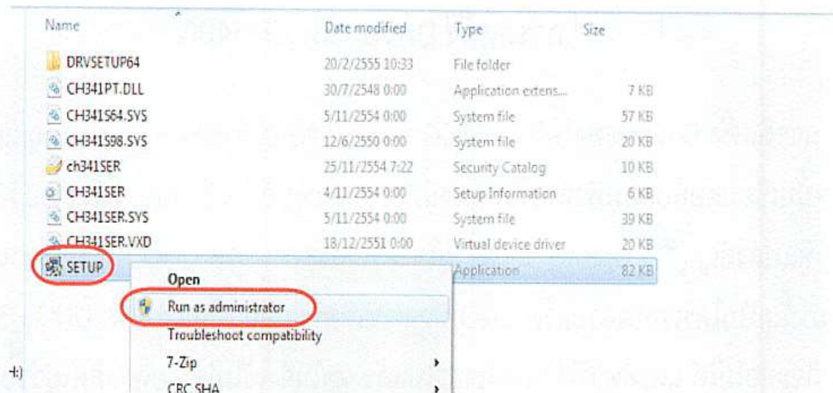
การติดตั้ง Driver ชิพ CH340G

การติดตั้ง Driver ของไอซี CH340G ซึ่งเป็นชิพที่ใช้สำหรับการแปลงสัญญาณ USB to Serial เพื่อให้บอร์ดนั้นเชื่อมต่อกับคอมพิวเตอร์ได้ ตรงนี้ต้องทำความเข้าใจกันก่อน ว่าไดรเวอร์นั้นเป็นไดร์เวอร์ของชิพที่ทำหน้าที่แปลงสัญญาณ USB to Serial ไม่ใช่ไดร์เวอร์ของบอร์ด UNO หลายๆ คนอาจจะเข้าใจว่า การติดตั้งไดร์เวอร์นั้นเป็นไดร์เวอร์ของบอร์ด UNO ปัญหาที่เกิดขึ้นก็คือ เมื่อนำบอร์ด UNO R3 ที่ใช้ชิพคนละเบอร์มาเชื่อมต่อ ก็จะต่อไม่ได้ และคิดว่าได้ลงไดร์เวอร์ไปแล้ว ทำไมจึงใช้ไม่ได้เลยพาลคิดไปว่าบอร์ดนั้นเสียไปแล้วก็มี จึงให้เข้าใจไว้ตรงนี้เลยว่า ให้ดูที่ชิพแปลงสัญญาณเป็นหลัก ว่าบอร์ดนั้นใช้ชิพเบอร์อะไร บางรุ่นอาจจะใช้ไอซีเบอร์ FT232 และในบางรุ่นก็อาจเป็นชิพของ ATMEL ATmega8 ก็มี แล้วแต่การออกแบบในแต่ละรุ่น ดังนั้นจึงอย่ายึดติดกับไดรเวอร์มากนัก ให้ดูจากบอร์ดที่ใช้ว่าใช้ไอซีในการเชื่อมต่อเบอร์อะไร แต่บอร์ดที่เราใช้ในหนังสือเล่มนี้คือ UNO R3 SMD ใช้ชิพในการแปลงสัญญาณ เบอร์ CH340G หลังจากได้หาดาวน์โหลดไดรเวอร์มาได้แล้ว ให้ทำการแยกไฟล์ออกมาก่อน



ให้ทำการแยกไฟล์ออกมาก่อน





มองหาไฟล์ SETUP หากเป็นวินโดว์ 7 ขึ้นไปให้ใช้วิธี คลิกขวา

แล้วเลือกไปที่ Run as administrator

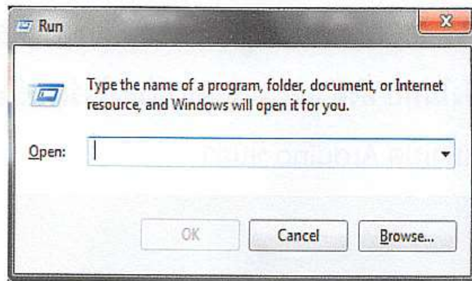


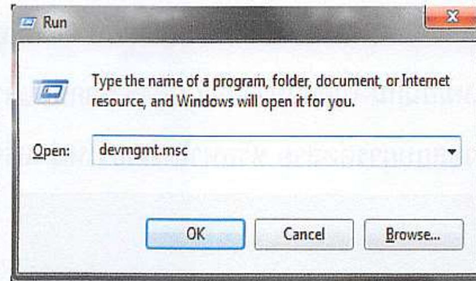
เสร็จแล้วคลิก OK

ติดตั้งเสร็จแล้วให้ไปตรวจสอบว่าบอร์ดของเรานั้นต่ออยู่กับ Port ไหน ต้องเสียบบอร์ดเข้ากับคอมพิวเตอร์ก่อน แล้วทำการเปิด Device Manager

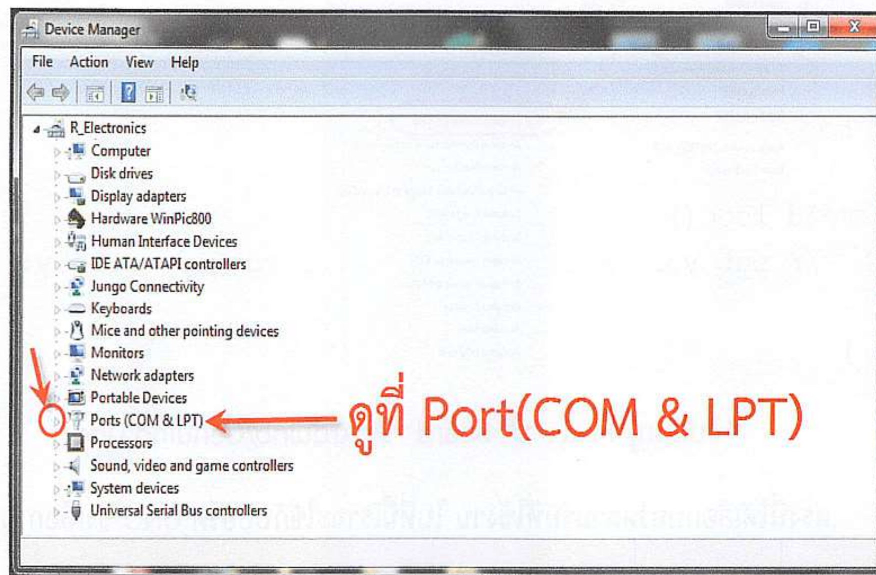


จะได้หน้าต่าง Run ขึ้นมาตามรูป

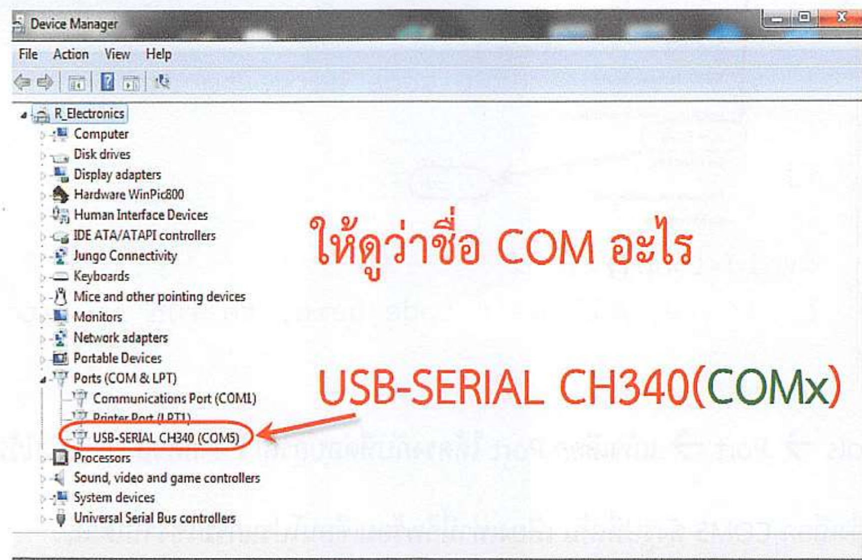




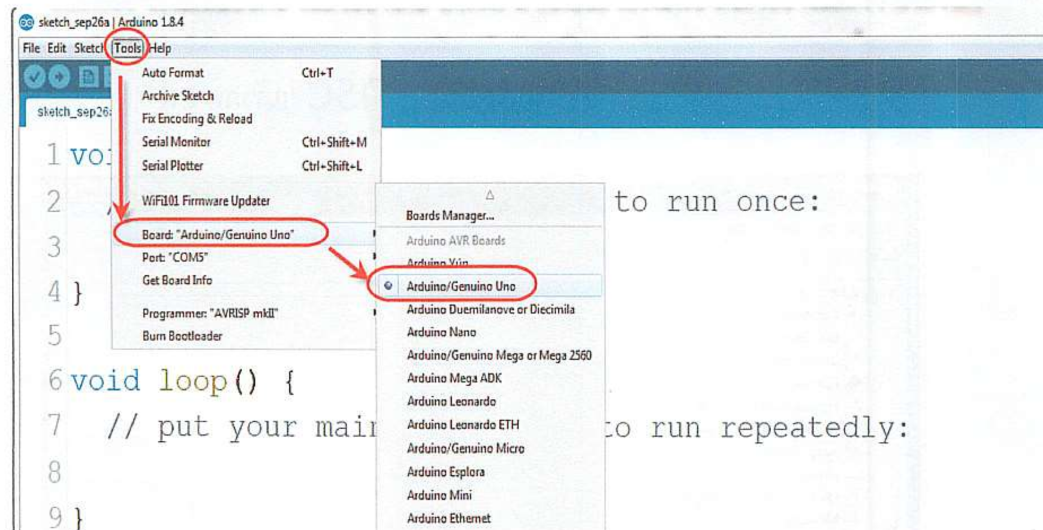
พิมพ์คำสั่ง **devmgmt.msc** แล้วกด OK



คลิกที่รูปสามเหลี่ยมเล็กๆ หน้า Port(COM & LPT)

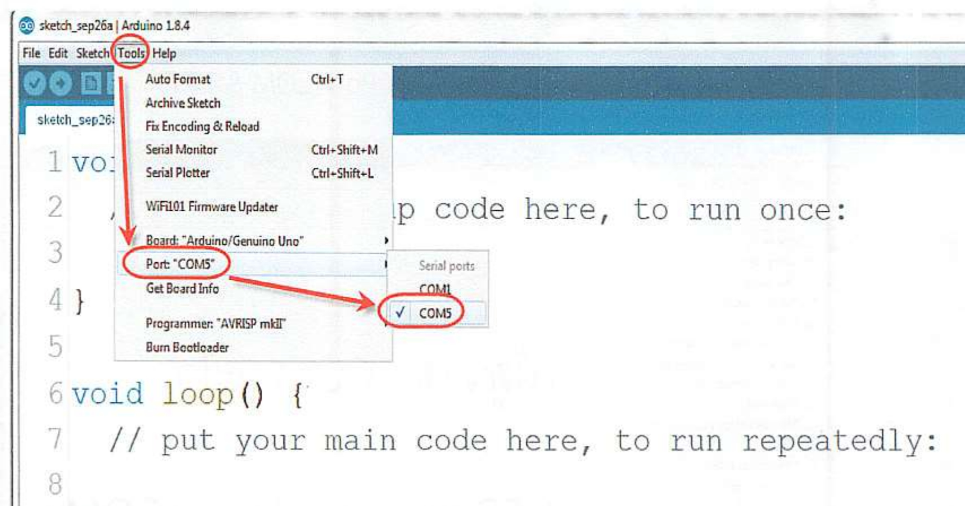


ให้ดูที่ USB-SERIAL CH340(COM) ว่าต่ออยู่กับ COM อะไร ชื่อCOM ตรงนี้อยู่ที่การเชื่อมต่อของแต่ละเครื่อง จะไม่เหมือนกัน ในตัวอย่างคือ COM5 หากต่อแล้วยังไม่พบ ดังตัวอย่างแสดงว่ายังลงไดร์เวอร์ไม่สำเร็จ ให้ตรวจสอบการต่อสาย หรือหาไดร์เวอร์ใหม่ เสร็จแล้วเปิด Arduino ขึ้นมา



ไปที่เมนู Tools → Board → Arduino/Genuino Uno

ตรงนี้ให้เลือกบอร์ดตามรุ่นที่ใช้งาน ในที่นี้เราจะใช้กับบอร์ด UNO จึงเลือก UNO



ไปที่ Tools → Port → แล้วเลือก Port ให้ตรงกับที่ต่อบอร์ดไว้ จากตัวอย่างได้ต่อไว้ที่ Port COM5 จึงเลือก COM5 ดังรูปที่เห็น เพียงเท่านี้ก็พร้อมเขียนโปรแกรมใช้งานได้แล้ว

ส่วนบอร์ด UNO รุ่นอื่นที่ไม่ได้ใช้ชิพ CH340G นั้น ทาง Arduino ได้มีไดร์เวอร์มาให้แล้วโดยไดร์เวอร์ทั้งหมด จะอยู่ในโฟลเดอร์โปรแกรมของ Arduino และอยู่ในโฟลเดอร์ที่ชื่อว่า drivers

ไดรเวอร์ชิพเบอร์อื่นๆ ที่อยู่ในโฟลเดอร์ Arduino

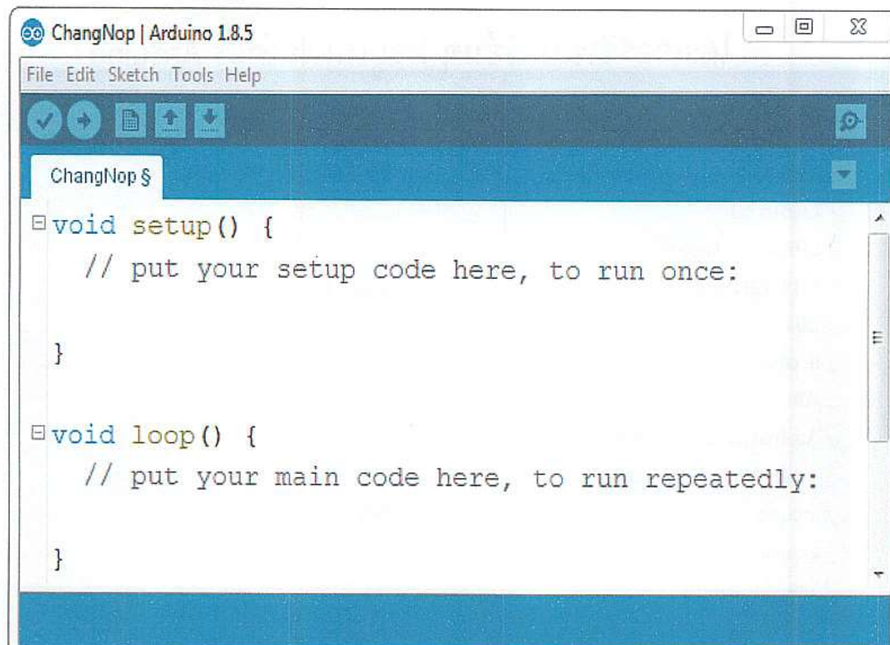
Name	Date modified	Type	Size
amd64	14/10/2560 23:43	File folder	
CP210x_6.7	14/10/2560 23:43	File folder	
CP210x_6.7.4	14/10/2560 23:43	File folder	
FTDI USB Drivers	14/10/2560 23:43	File folder	
ia64	14/10/2560 23:43	File folder	
license	14/10/2560 23:43	File folder	
x86	14/10/2560 23:43	File folder	
AdafruitCircuitPlayground	2/10/2560 20:37	Security Catalog	10 KB
AdafruitCircuitPlayground	2/10/2560 20:37	Setup Information	4 KB
arduino	2/10/2560 20:37	Security Catalog	11 KB
arduino	2/10/2560 20:37	Setup Information	10 KB
arduino_gemma	2/10/2560 20:37	Security Catalog	11 KB
arduino_gemma	2/10/2560 20:37	Setup Information	8 KB
arduino-org	2/10/2560 20:37	Security Catalog	10 KB
arduino-org	2/10/2560 20:37	Setup Information	11 KB
dpinst-amd64	2/10/2560 20:37	Application	1,024 KB
dpinst-x86	2/10/2560 20:37	Application	901 KB
genuino	2/10/2560 20:37	Security Catalog	9 KB
genuino	2/10/2560 20:37	Setup Information	5 KB
linino	2/10/2560 20:37	Setup Information	4 KB
linino-boards_amd64	2/10/2560 20:37	Security Catalog	7 KB
linino-boards_x86	2/10/2560 20:37	Security Catalog	7 KB
Old_Arduino_Drivers	2/10/2560 20:37	เอกสาร WinRAR แ...	17 KB
README	2/10/2560 20:37	Text Document	1 KB

การใช้งานโปรแกรม Arduino IDE

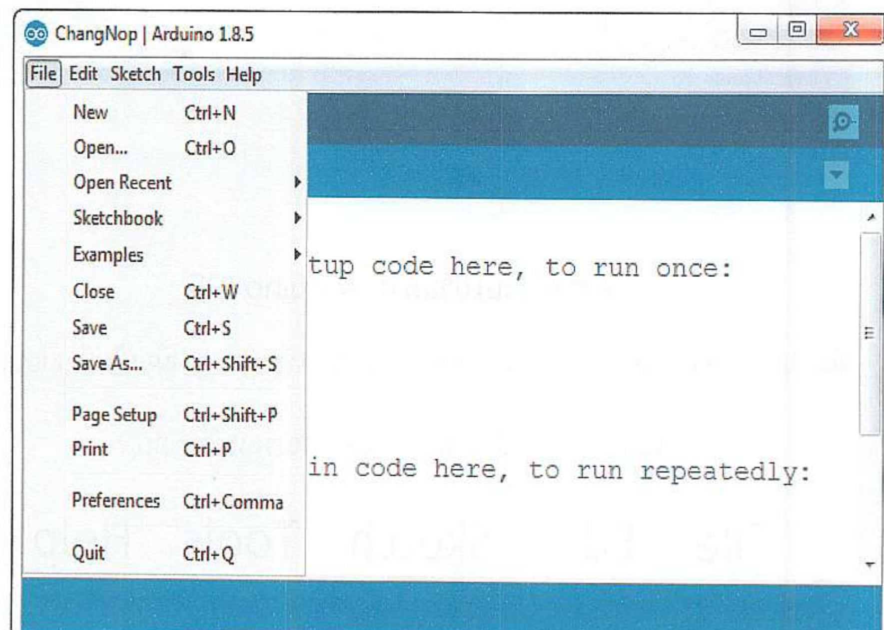
เมื่อเปิดโปรแกรมขึ้นมาจะพบกับหน้าตาโปรแกรมที่ดูเรียบง่ายดูเหมือนไม่มีอะไรซับซ้อน

ที่เมนูบาร์ นั้น มีเมนูอยู่ทั้งหมด 5 หมวดหมู่ด้วยกัน

File Edit Sketch Tools Help



ในส่วนของเมนู File นั้นได้มีเมนูย่อยทั้งหมด 12 เมนูด้วยกัน เมื่อคลิกที่ File จะปรากฏเมนูย่อยออกมา



New ใช้สำหรับการสร้างหน้าต่างงานใหม่ขึ้นมาโดยโปรแกรมจะไม่ปิดงานเดิม

Open ใช้สำหรับเปิดงานที่มีอยู่แล้วในเครื่อง

Open Recent ใช้สำหรับเปิดงานที่เคยเปิดทำครั้งล่า สามารถแสดงได้ 10 งาน

Sketchbook ใช้เปิดงานที่บันทึกไว้ในโฟลเดอร์มาตรฐาน โดยสามารถกำหนดการตั้งค่าได้ที่เมนู Preferences

Examples ใช้เพื่อเปิดดูโค้ดตัวอย่างการเขียนโปรแกรม และ การใช้งานไลบรารี่ต่างๆ ที่มีมาให้

Close ปิดหน้าต่างโปรแกรม เฉพาะหน้าต่างที่ต้องการปิด

Save ใช้บันทึกงานที่กำลังทำ

Save As ใช้บันทึกงานที่กำลังทำไปเก็บในไดร์ หรือ โฟลเดอร์ ที่เราระบุตำแหน่งเองได้

Page Setup ใช้ตั้งค่าหน้ากระดาษเพื่อการพิมพ์

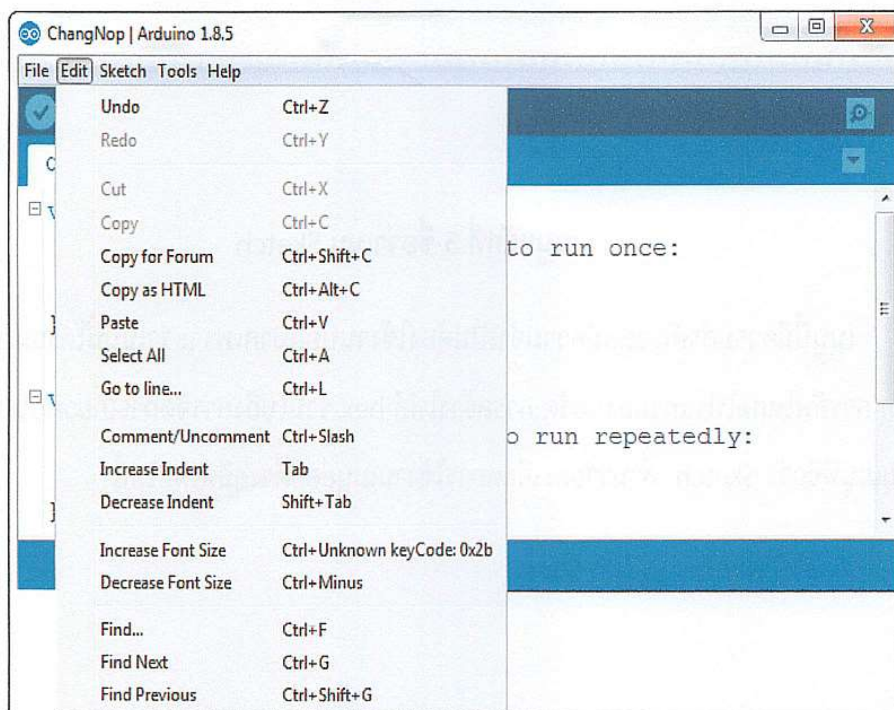
Print ใช้สั่งพิมพ์โค้ดโปรแกรมที่เปิดอยู่ออกมาเป็นเอกสาร

Preferences ใช้สำหรับการตั้งค่าต่างๆ ของโปรแกรม

Quit ออกจากโปแกรมทั้งหมด หากเปิดไว้หลายๆงานโปรแกรมจะถูกปิดลงทั้งหมด

ถัดมาเป็นเมนูที่ชื่อว่า Edit มีเมนูย่อยทั้งหมด 15 เมนูด้วยกัน

ส่วนของเมนูนี้ไม่ค่อยมีอะไรมากนักการใช้งานก็จะคล้ายๆกับโปรแกรมทั่วไป ใช้สำหรับแก้ไขการเขียนโค้ด



Undo ใช้สำหรับคืนค่าเดิมของงาน เช่น เราสั่งลบโค้ดที่เขียนออกไป การใช้คำสั่ง Undo จะช่วยคืนกลับมา

Redo จะช่วยย้อนกลับไปก่อนที่จะ Undo และจะต้องมีการกดปุ่ม Undo แล้วเท่านั้นถึงจะ Redo ได้

Cut ใช้สำหรับตัดข้อความหรือโค้ดใดใดออกไปเพื่อเอาไปวางอีกที่หนึ่ง

Copy ใช้สำหรับคัดลอกข้อความหรือโค้ดใดใด เพื่อเอาไปวางอีกที่หนึ่ง

Paste ใช้วางข้อความหรือโค้ดใดใด ที่มาจากคำสั่ง cut หรือ copy

Select All ใช้เลือกตัวอักษรทั้งหมด

Go to Line... ใช้เพื่อเลือกไปที่บรรทัดที่กำหนด โดยจะมีหน้าต่างขึ้นมาให้ใส่หมายเลขบรรทัด

Comment/Uncomment ใช้เพื่อใส่เครื่องหมาย // เพื่อให้บรรทัดนั้นเป็นคอมเม้น หรือ ยกเลิกก็ได้

Increase Indent ใช้เลื่อนข้อความหรือโค้ดไปทางขวา

Decrease Indent ใช้เลื่อนข้อความหรือโค้ดไปทางซ้าย

Find... ใช้สำหรับค้นหา ตัวอักษร คำ หรือ ข้อความ หรือชื่อตัวแปรต่างๆในโปรแกรม

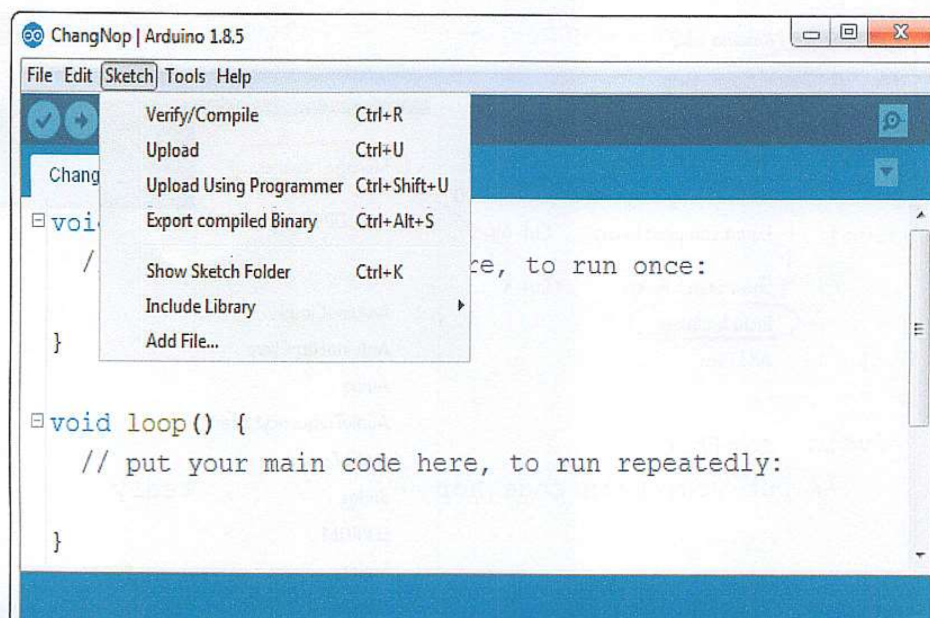
Find Next ใช้สำหรับค้นหา จาก Find... ค้นหาไปเรื่อยๆทีละตัว

Find Previous เหมือนกับ Find Next แต่จะเลื่อนย้อนกลับหลังทีละตัว

****

เมนูหลักที่ 3 ชื่อว่าเมนู Sketch

เมนูนี้มีความสำคัญและมีความจำเป็นต้องใช้งานบ่อยมากเพราะว่าเมนูนี้ได้รวมเอาการ คอมไพล์โปรแกรม การอัพโหลดโปรแกรมลงบอร์ด การสร้างไฟล์ hex รวมไปถึงการจัดการเกี่ยวกับไลบรารีทั้งหมด ได้รวมอยู่ในเมนูที่ชื่อว่า Sketch ส่วนรายละเอียดการใช้งานเมนูย่อยต่างๆมีดังต่อไปนี้



Verify/Compile ใช้เพื่อตรวจสอบและคอมไพล์โปรแกรมที่เขียนขึ้นมา

Upload ใช้สำหรับอัพโหลดโปรแกรมลงบอร์ดได้โดยตรง โดยต่อผ่านพอร์ต USB

Upload Using Programmer ใช้สำหรับอัพโหลดโปรแกรมลงบอร์ดโดยใช้เครื่องโปรแกรม

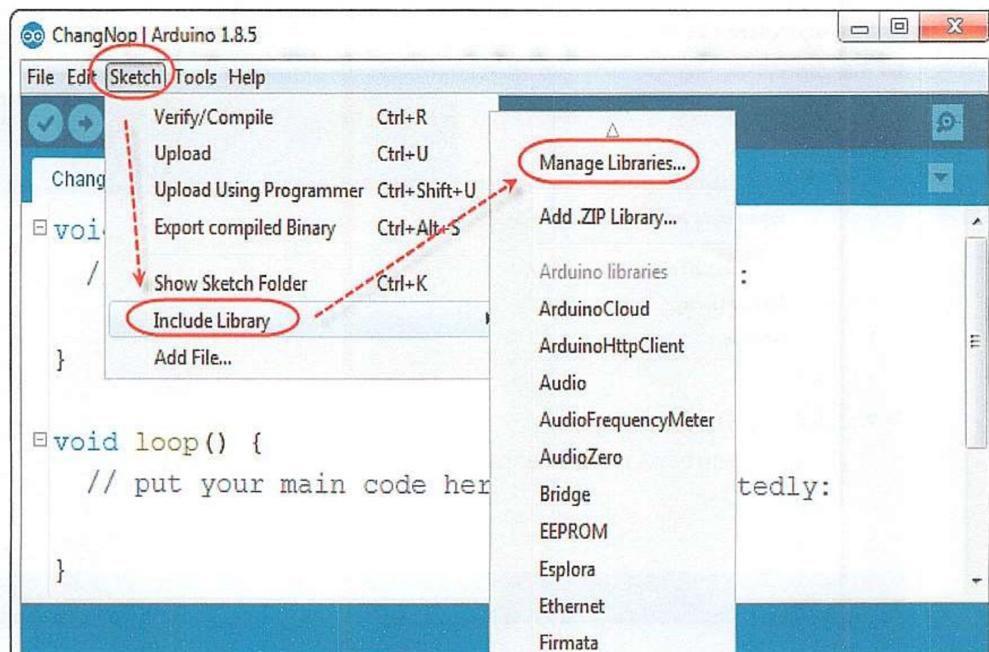
Export compiled Binary ใช้สำหรับสร้างไฟล์ hex โดยจะสร้างไปเก็บไว้ในโฟลเดอร์ที่เก็บงาน

Show Sketch Folder ใช้สำหรับเปิดโฟลเดอร์ที่เก็บงาน

Include Library ใช้สำหรับโหลดไลบรารี หรือ เพิ่มเข้ามาจากแหล่งที่กำหนดจากไฟล์ ZIP

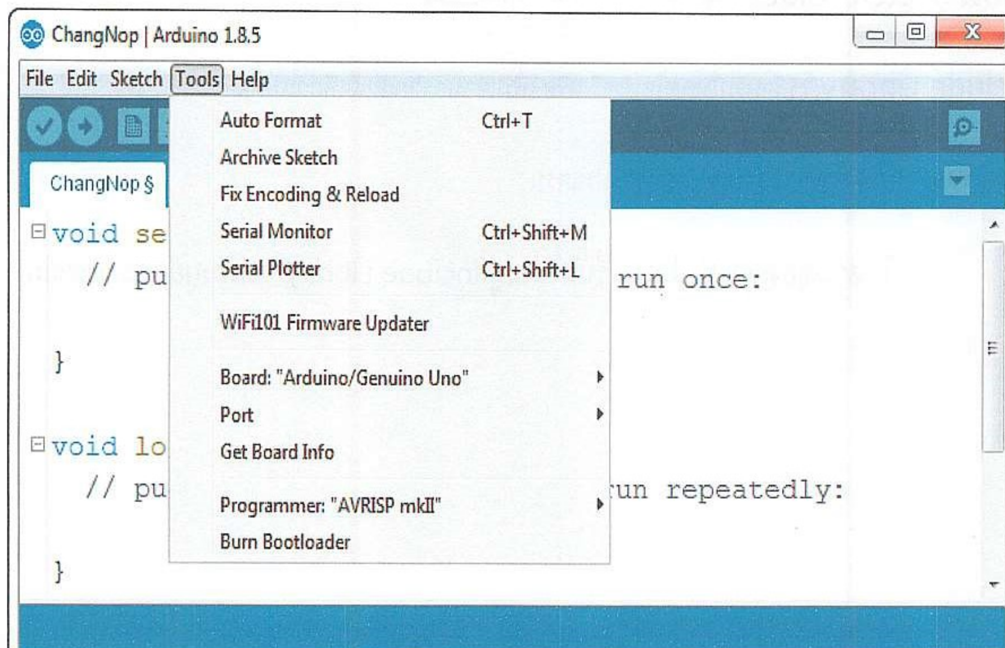
Add File... ใช้สำหรับเพิ่มไฟล์เข้ามาในโปรแกรม

ในส่วนของการเพิ่มไลบรารี่นั้นในเมนู Include Library จะมีเมนูย่อยออกมาอีก

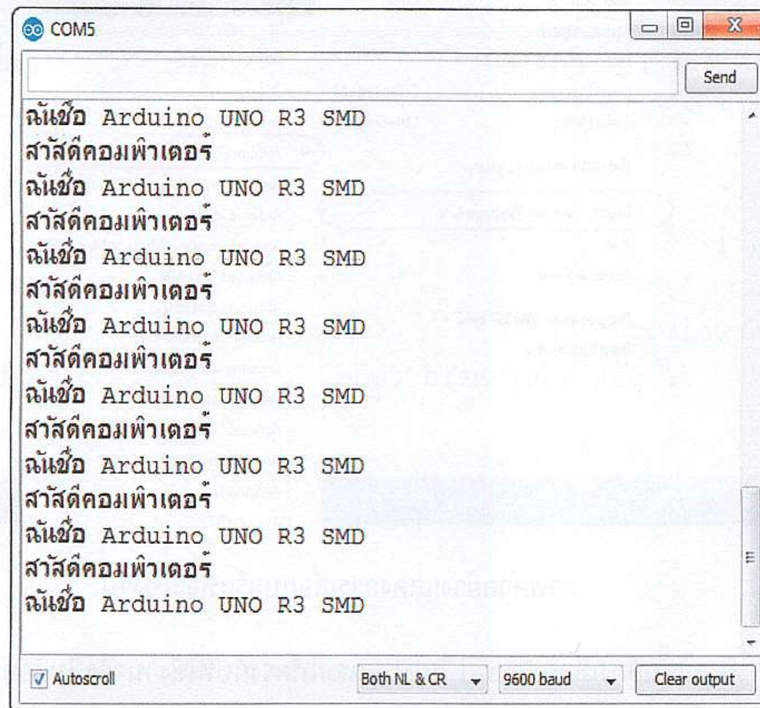


เมนูหลัก เมนูที่ 4 ชื่อว่า Tools มีเมนูย่อยทั้งหมด 11 เมนูด้วยกัน

เมนูนี้เป็นเมนูที่สำคัญมากอีกเมนูหนึ่งเพราะได้รวมเอาเครื่องมือ และ การตั้งค่าบอร์ด รวมไปถึงการตั้งค่าพอร์ตในการเชื่อมต่อกับบอร์ดที่เราจะใช้เขียนโปรแกรมนั่นเอง

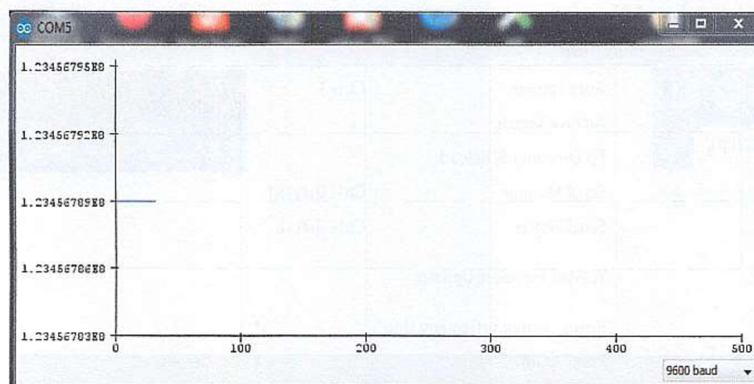


Serial Monitor เปิดโปรแกรม Serial Monitor แสดงข้อมูลที่ส่งมาจากบอร์ด Arduino

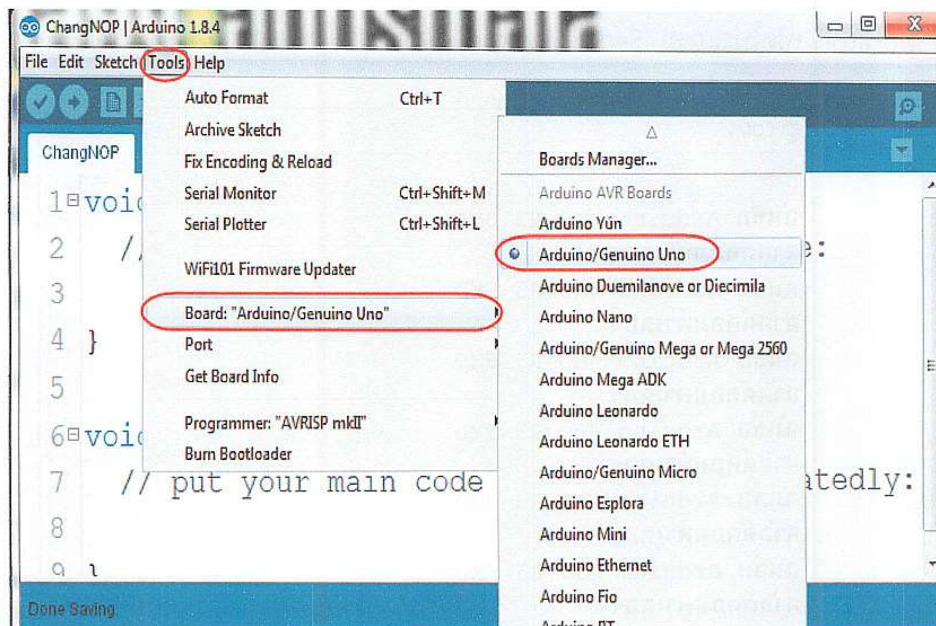


ภาพตัวอย่าง

Serial Plotter แสดงข้อมูลเป็นรูปแบบเส้นกราฟ



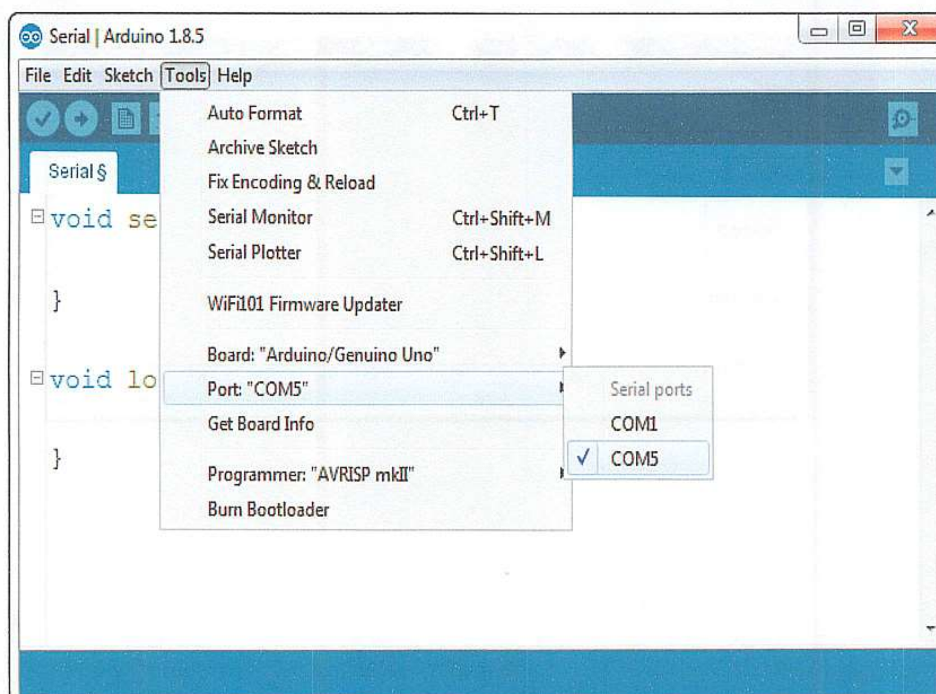
Board: xxx ใช้สำหรับเลือกรุ่นบอร์ดที่จะใช้งาน (ต้องเลือกให้ตรงกับบอร์ดจริงที่ใช้)



ภาพตัวอย่างแสดงการเลือกบอร์ดที่จะใช้งาน

ในการใช้งานนั้นต้องเลือก Board ในโปรแกรมให้ตรงกับที่ใช้งานจริง ในตัวอย่างคือ UNO

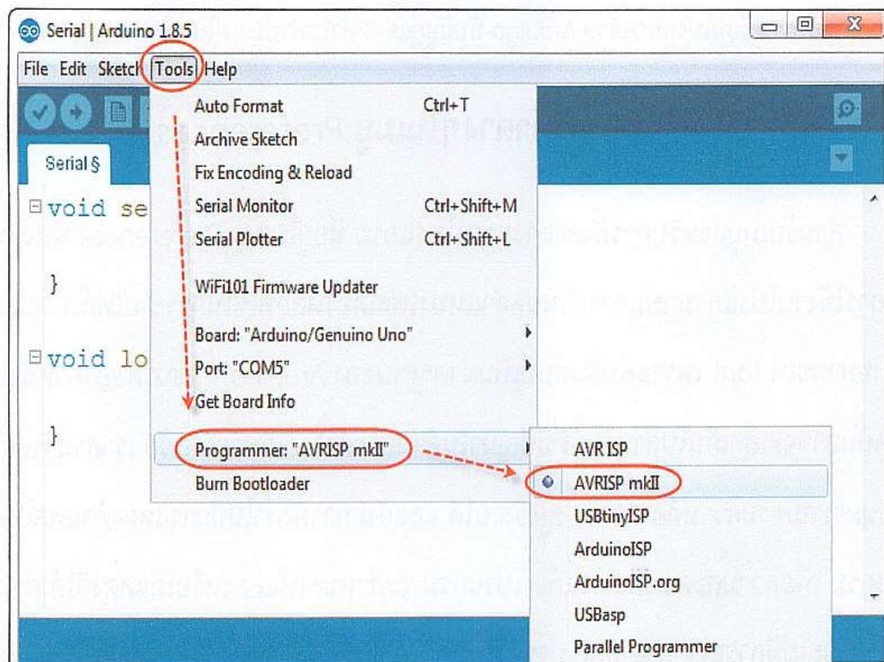
Port ใช้เลือกพอร์ตที่จะเชื่อมต่อกับบอร์ด (ต้องเลือกให้ตรงไม่เช่นนั้นจะอัพโหลดโปรแกรมเข้าบอร์ดไม่ได้)



ภาพตัวอย่างแสดงการเลือก Port ที่จะเชื่อมต่อกับคอมพิวเตอร์

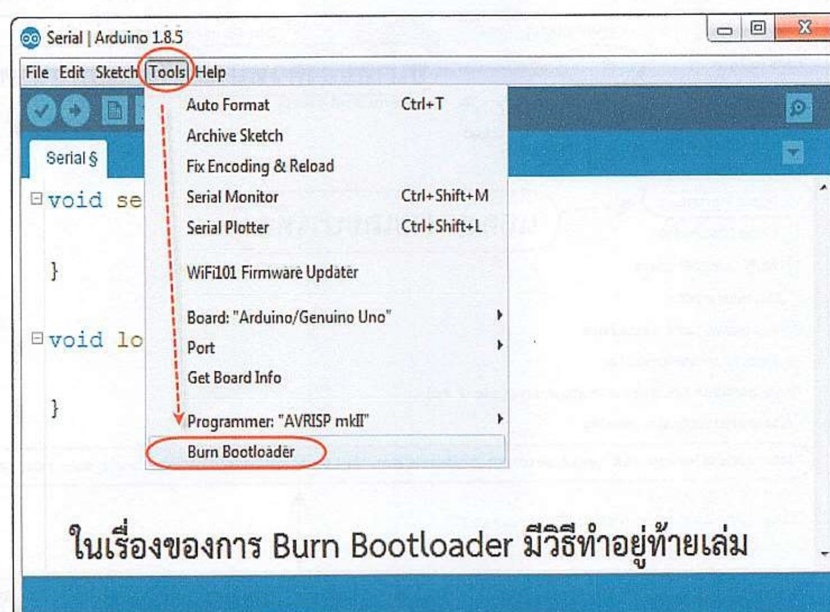
Get Board Info ใช้สำหรับดูรายละเอียดของบอร์ดที่เชื่อมต่ออยู่กับโปรแกรม

Programmer ใช้สำหรับเลือกเครื่องโปรแกรมที่จะใช้เบิร์นไฟล์โปรแกรมเข้าสู่บอร์ด Arduino



ภาพแสดงการเลือก Programmer สำหรับการเบิร์นโปรแกรมลงบอร์ด Arduino

Burn Bootloader ความหมายตรงตัว คือเบิร์นไฟล์บู๊ตโหลดเดอร์เข้าตัวไอซี (จะใช้งานในกรณีที่ซื้อไอซีตัวใหม่ที่ยังไม่มีบู๊ตโหลดเดอร์มาใส่นั้นจะต้องทำการ Burn bootloader เข้าไปเสียก่อน)

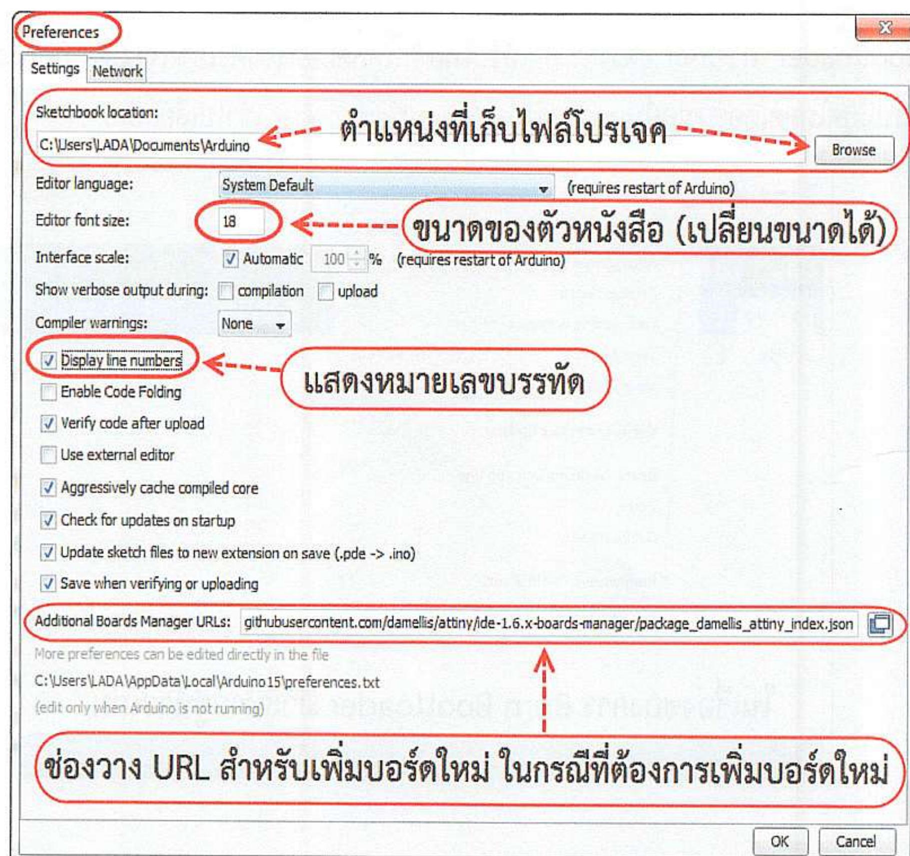


หากจะเขียนโปรแกรมควบคุมสั่งงานบอร์ดไมโครคอนโทรลเลอร์สำเร็จรูป ที่เราเรียกมันว่า Arduino จำเป็น จะต้องทำความรู้จักกับโครงสร้างการในการเขียนโปรแกรมด้วย Arduino กันก่อน

โครงสร้างหลักของการเขียนโปรแกรมด้วย Arduino นั้นมีอยู่แค่ 3 ส่วนเท่านั้น แต่ในแต่ละส่วนนั้น จะมีหัวข้อย่อยลงไปอีก

การตั้งค่าต่างๆในเมนู Preferences

สำหรับเมนูสำหรับการตั้งค่าต่างๆของโปรแกรม ที่อยู่ในเมนู Preferences นั้นจะรวมการตั้งค่าที่เกี่ยวกับการใช้งานโปรแกรม เช่นการกำหนดตำแหน่งโฟลเดอร์หลักที่ใช้ในการเก็บไฟล์โปรแกรมที่เขียนขึ้น การตั้งค่าขนาดของ font เพราะค่าเริ่มต้นที่เป็นมาตรฐานของ Arduino นั้นขนาดของตัวหนังสือที่ใช้ในการเขียนโค้ดนั้นอาจจะเล็กเกินไปสำหรับผู้ที่มีปัญหาในด้านสายตาโดยเฉพาะผู้สูงวัย เราสามารถตั้งค่าขนาดของ font ให้เหมาะสมตามความต้องการของผู้ใช้งานได้ และยังมีการตั้งค่าให้มีการแสดงตัวเลขจำนวนบรรทัดของโค้ดโปรแกรม เพื่อความสะดวกในการแก้ไขโปรแกรม จะกำหนดให้แสดงหรือไม่แสดงก็ได้ ค่าเริ่มต้นของโปรแกรมนั้นจะไม่มีการแสดงหมายเลขบรรทัด หากต้องการให้แสดงหมายเลขบรรทัด ก็เพียงแค่ใส่เครื่องหมายลงในช่องหน้า Display line Numbers หากไม่ต้องการให้แสดงก็เพียงแค่เอาเครื่องหมายออกเท่านั้นเอง ส่วนการตั้งค่าอื่นๆก็ทำได้ด้วยวิธีเดียวกัน



หลักของการเขียนโปรแกรม Arduino

หลักของการเขียนโปรแกรมด้วย Arduino นั้นมีแค่ 3 หัวข้อใหญ่เท่านั้น จำให้ดี

Structure โครงสร้างต่างๆของโปรแกรม

โครงสร้าง และ สัญลักษณ์ตัวอักษรพิเศษ รวมถึงเครื่องหมาย ตัวดำเนินการทางคณิตศาสตร์

Variables ตัวแปร และ ค่าคงที่

ตัวแปร และ ค่าคงที่ต่าง ๆ ที่ใช้ในการเก็บข้อมูล และประมวลผลหรือการคำนวณต่าง ๆ

Functions ฟังชั่นสำเร็จรูปที่ Arduino มีให้ใช้

ฟังก์ชันสำเร็จรูปที่ Arduino สร้างมาให้ใช้แบบง่ายๆ สบายๆ

ในหมวดหมู่แรกคือ Structure หรือ โครงสร้างนั้น โครงสร้างหลักที่จำเป็นที่สุดคือ ฟังก์ชันหลักของ Arduino มีอยู่ด้วยกัน 2 ฟังก์ชันเท่านั้น

เมื่อทำการเปิดโปรแกรมขึ้นมาโปรแกรมจะสร้างฟังก์ชันหลักทั้งสองฟังก์ชันนี้ขึ้นมาให้เองโดยอัตโนมัตินั่นคือ ฟังก์ชันที่มีชื่อว่า setup และ ฟังก์ชัน loop ในการเขียนโปรแกรมด้วยภาษา C นั้นจะต้องทำความเข้าใจในเรื่องของฟังก์ชันให้เข้าใจ เพราะฟังก์ชันเป็นสิ่งที่สำคัญมาก ผู้เขียนจะอธิบายถึงสิ่งสำคัญและรายละเอียดต่าง ๆ ที่จำเป็นเกี่ยวกับการเขียนโปรแกรมด้วยภาษา C ในบทถัดไปในหนังสือเล่มนี้ขอให้ผู้อ่านค่อยๆทำความเข้าใจเรื่องราวต่าง ๆ ในเบื้องต้นอย่างละเอียด เพราะพื้นฐานนั้นเป็นสิ่งสำคัญมาก

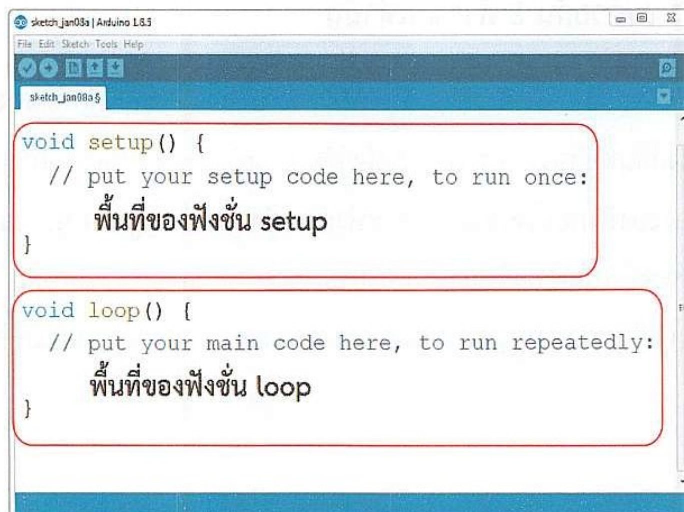
ฟังก์ชัน setup()

```
void setup(){
  //ใช้เขียนคำสั่งสำหรับการตั้งค่าต่างๆ
  //จะทำงานเพียงรอบเดียวเท่านั้น
}
```

ฟังก์ชัน loop()

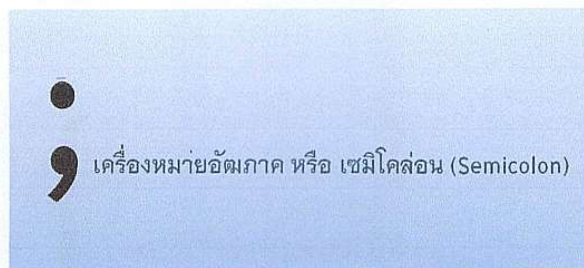
```
void loop(){
  //ใช้เขียนคำสั่งที่จะให้ทำงานตลอดเวลา
  //จะทำงานซ้ำแล้วซ้ำอีก วนอยู่ในนี้
  //ยกเว้นคำสั่งในนี้จะสั่งให้โดดออกไปทำงานข้างนอก
  //เสร็จแล้วโปรแกรมก็จะกลับมาทำงานในนี้ต่อไป อยู่อย่างนี้
}
```

นี่เป็นฟังก์ชันหลักที่เปิดโปรแกรมขึ้นมา Arduino ก็จะสร้างขึ้นมาให้เราเองโดยอัตโนมัติ เราเพียงแค่เขียนคำสั่งลงไปในพื้นที่ ที่ใช้เขียนคำสั่ง เท่านั้น แต่เรายังสร้างฟังก์ชันขึ้นมาใช้งานเองได้อีกด้วย แต่การตั้งชื่อฟังก์ชันนั้นต้องไม่ซ้ำกันกับฟังก์ชันที่มีอยู่



ภาพตัวอย่างโครงสร้าง ฟังก์ชันหลักของจริง ที่ใช้งานจริง

ฟังก์ชัน setup นั้นมีไว้สำหรับการตั้งค่าพอร์ตต่าง ๆ ในบอร์ด ว่าเราจะใช้งานขาใดบ้าง และ จะใช้ขาใดทำงานเป็นอินพุต หรือ จะใช้ขาใดทำงานเป็นเอาต์พุต เนื่องจากขาของไมโครคอนโทรลเลอร์นั้นมันสามารถที่จะทำงานเป็นอินพุต เพื่อใช้รับสัญญาณเข้ามา หรือจะทำงานเป็นเอาต์พุตเพื่อใช้ส่งสัญญาณออกไปก็ได้ ยกตัวอย่างเช่น เราต้องการใช้ขา 7 เป็นอินพุตรับสัญญาณจากการกดสวิตช์ และใช้ขา 8 เป็นเอาต์พุตเพื่อให้ทำงานขับหลอด LED หรือเราอยากจะใช้ ขา 8 เป็นอินพุตรับการกดสวิตช์ แล้วให้ขา 7 เป็นเอาต์พุตส่งสัญญาณออกไปขับหลอด LED ก็ทำได้เช่นกัน นอกจากการตั้งค่าสถานะ การทำงานของแต่ละขาได้แล้ว การตั้งค่าความเร็วในการรับ-ส่งข้อมูล เมื่อนำไปเชื่อมต่อกับสิ่งอื่น ๆ เช่น คอมพิวเตอร์ Bluetooth ,WIFI หรือโมดูลสื่อสารต่าง ๆ ทั้งหมดที่กล่าวมาทำได้ด้วยการเขียนคำสั่งลงไปในฟังก์ชันที่มีชื่อว่า **setup** นั่นเอง แต่ฟังก์ชันนี้โปรแกรมจะทำงานเพียงรอบเดียวเท่านั้น เสร็จแล้วโปรแกรมจะเข้าทำงานในฟังก์ชัน **loop** และทำงานวนอยู่อย่างนั้น จนกว่าจะหยุดจ่ายไฟ หรือมีการรีเซ็ตการทำงาน ดังนั้นการเขียนคำสั่งที่ต้องการจะให้มันทำงานซ้ำ ๆ วน ๆ ให้ทำงานตลอดเวลา จึงต้องมาเขียนคำสั่งไว้ในฟังก์ชัน **loop** นั่นเอง และ สิ่งสำคัญอีกอย่างหนึ่ง สำหรับการเขียนโปรแกรมด้วยภาษา C,C++ นั้นก็คือต้องรู้จักตัวอักษร และสัญลักษณ์ต่าง ๆ ที่ใช้งานในภาษา C หากผู้อ่านเคยเห็นโค้ดโปรแกรมต่าง ๆ ในภาษา C มาก่อนจะเห็นว่าในโค้ดต่าง ๆ จะมีตัวอักษร และสัญลักษณ์แปลกๆอยู่เต็มไปหมดซึ่งจริง ๆ แล้วมันก็มีอยู่ไม่กี่ตัว เราเพียงแค่ต้องรู้ว่าอักษรหรือสัญลักษณ์แปลกๆ ที่เราไม่เคยได้ใช้ในชีวิตประจำวันเลยนั้น เขามีไว้ทำอะไร และต้องเขียนลงไปตรงไหน และ มันมีความหมายว่าอะไร

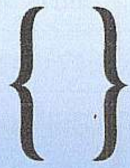


รูปแบบการเขียนเครื่องหมายนี้ในภาษา C คือ เมื่อเขียนคำสั่งจบหนึ่งคำสั่งให้ใส่เครื่องหมายนี้ปิดท้าย เป็นการบอกว่าจบคำสั่งนี้แล้ว ห้ามลืมเด็ดขาด เพราะถ้าลืมจะไม่สามารถคอมไพล์โปรแกรมได้

ตัวอย่าง

```
pinMode(13,OUTPUT);
```

ภาพแสดงตัวอย่างการใช้งานเครื่องหมายเซมิโคลอน ปิดท้ายเมื่อเขียนจบ หนึ่งคำสั่ง



เครื่องหมายวงเล็บปีกกา หรือ เรียกว่าบล็อก (Block)

ในภาษา C นั้นจะมีการทำงานเป็นรูปแบบที่แยกการทำงานเป็นสัดส่วนเรียกว่า ฟังก์ชัน เครื่องหมาย { } ถูกนำมาใช้เพื่อเป็นการบ่งบอกถึงขอบเขตในการทำงานของแต่ละฟังก์ชัน เพราะในหนึ่งโปรแกรมนั้นอาจมีการทำงานหลายๆฟังก์ชัน ก็ได้ ดังนั้นในแต่ละฟังก์ชันจึงมีพื้นที่ในการทำงานของตัวเอง โดยมีเครื่องหมาย **{ }** เป็นตัวบอกเขตแดนว่าพื้นที่นั้น เป็นของฟังก์ชันใด



{ ปีกกาเปิด หมายถึง จุดเริ่มต้นการทำงานของฟังก์ชันนั้น



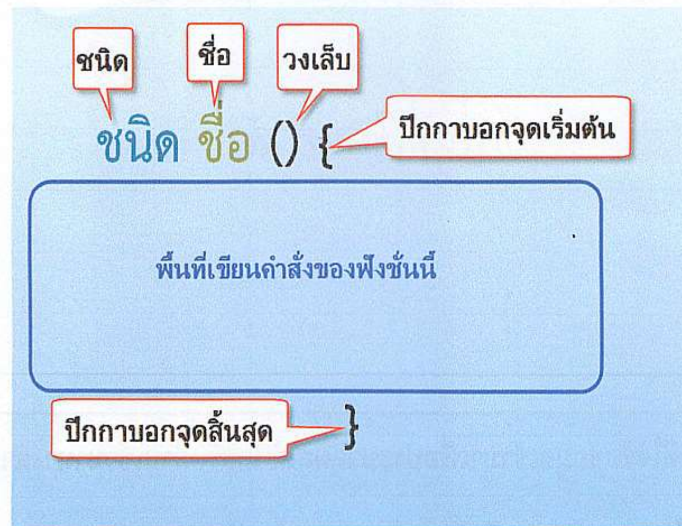
ปีกกาปิดหมายถึง จุดสิ้นสุดการทำงานของฟังก์ชันนั้น }

และคำสั่งที่จะเขียนสั่งงานจะต้องอยู่ภายในปีกกา หรือ บล็อก เท่านั้น หากเขียนคำสั่งนอกบล็อกจะคอมไพล์โปรแกรมไม่ผ่าน และต้องจำเอาไว้ว่ามีปีกกาเปิดเท่าไหร่ ก็ต้องต้องมีปีกกาปิดเท่ากัน

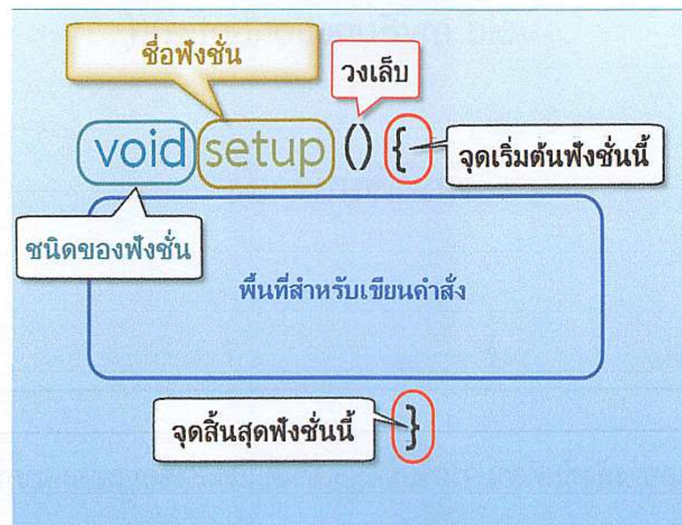
ฟังก์ชันนั้นจะต้องมีส่วนประกอบที่สำคัญ คือ ชนิดของฟังก์ชัน ชื่อฟังก์ชัน วงเล็บใช้สำหรับ รับ-ส่งข้อมูล กับฟังก์ชันอื่น ๆ และบล็อก **{ }** เพื่อบอกขอบเขตการทำงานของฟังก์ชันนั้น ๆ

โดยฟังก์ชันนั้นเราสามารถสร้างขึ้นเองได้แล้วแต่การทำงาน และ แต่ละฟังก์ชันสามารถมีการคืนค่า หรือ ส่งข้อมูลออกจากฟังก์ชันได้หลายชนิด เช่นเลขจำนวนเต็ม เลขที่มีจุดทศนิยม ตัวอักษร หรือไม่มีการส่งข้อมูลใดใดออกมาเลยก็ได้

ในการสร้างฟังก์ชันแบบไม่มีการส่งค่าข้อมูลใดใดออกจากฟังก์ชันเลย จะใช้ชื่อชนิดว่า **void** แล้วตามด้วยชื่อฟังก์ชันตรงนี้เราสามารถตั้งชื่อขึ้นมาเองได้ และควรตั้งชื่อให้สอดคล้องกับการทำงานของโปรแกรม เพื่อป้องกันความสับสน หรือหากต้องการสร้างฟังก์ชันที่ไม่มีการส่งข้อมูลออกจากฟังก์ชัน และไม่มีการรับข้อมูลใดใดเข้าฟังก์ชันเลย ก็เพียงแค่ใส่คำว่า void ลงในวงเล็บ

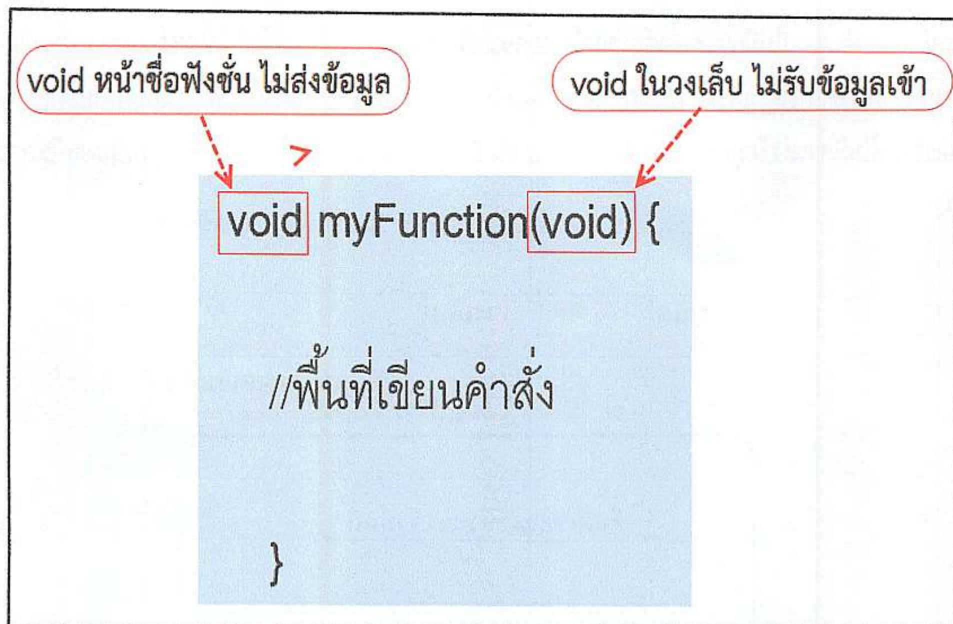


ภาพแสดงถึงโครงสร้างส่วนต่างๆของฟังก์ชัน

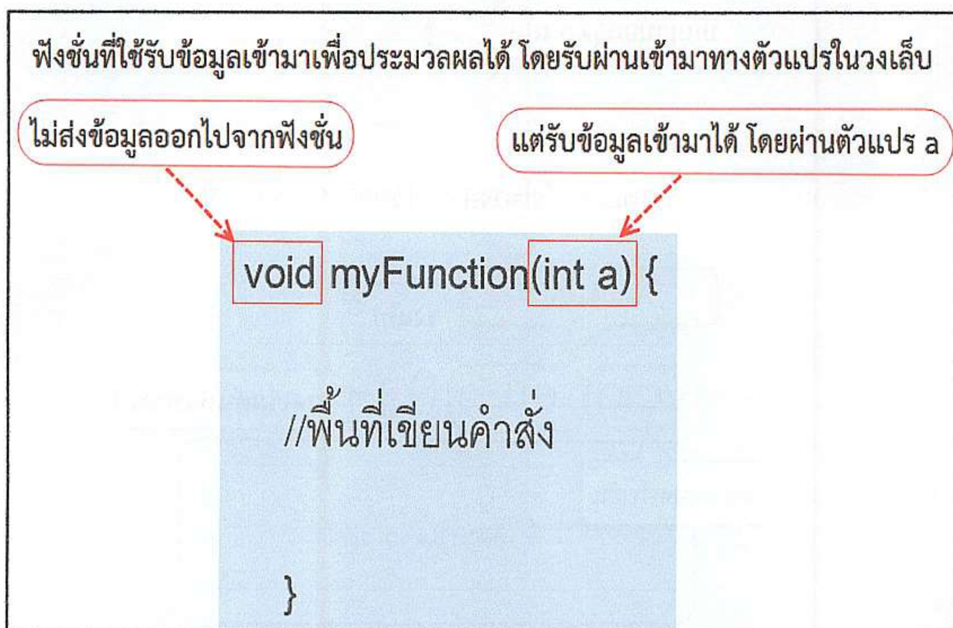


จากรูปตัวอย่างจะเห็นว่าเป็นฟังก์ชันที่ไม่มีการส่งค่าข้อมูลออกจากฟังก์ชัน โดยใช้ชื่อชนิดของฟังก์ชันว่า void และ ตั้งชื่อฟังก์ชันว่า setup หมายถึงการตั้งค่า บ่งบอกถึงหน้าที่การทำงาน ว่ามีไว้ใช้สำหรับการทำงานในการตั้งค่าต่าง ๆ

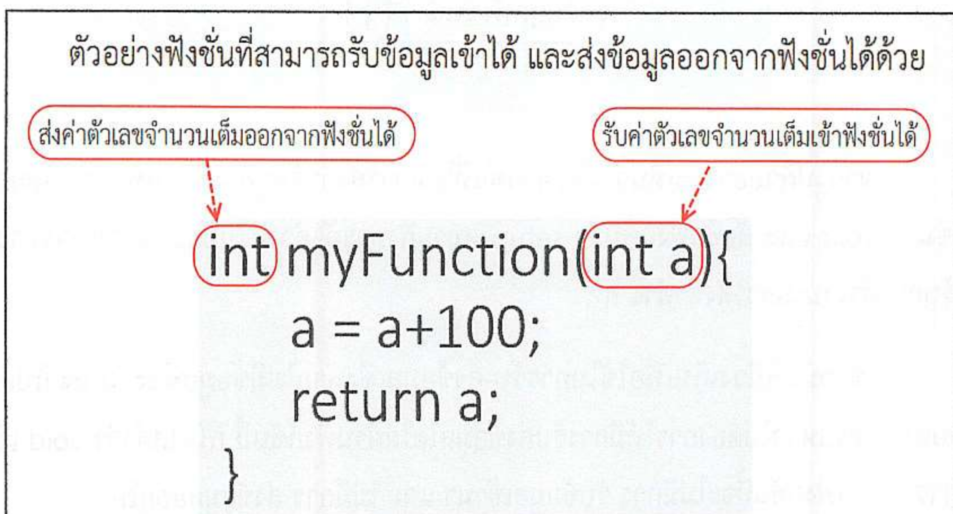
จากนั้นก็มีวงเล็บเพื่อใช้ในการรับ-ส่งข้อมูล ซึ่งหากไม่มีข้อมูลที่จะรับ-ส่ง ก็ปล่อยว่างไว้เป็นวงเล็บเปล่า หรือหากไม่ต้องการให้มีการรับส่งข้อมูลใดใดผ่านฟังก์ชันนี้ ก็ให้ใส่คำว่า void เข้าไปในวงเล็บ เป็นการบอกว่าฟังก์ชันนี้จะไม่มีการ รับข้อมูลเข้ามา และ ไม่มีการ ส่งข้อมูลออกไป



void หน้าชื่อฟังชั่น ไม่ส่งข้อมูล
void ในวงเล็บ ไม่รับข้อมูลเข้า
void myFunction(void) {
//พื้นที่เขียนคำสั่ง
}



ฟังชั่นที่ใช้รับข้อมูลเข้ามาเพื่อประมวลผลได้ โดยรับผ่านเข้ามาทางตัวแปรในวงเล็บ
ไม่ส่งข้อมูลออกไปจากฟังชั่น
แต่รับข้อมูลเข้ามาได้ โดยผ่านตัวแปร a
void myFunction(int a) {
//พื้นที่เขียนคำสั่ง
}



ตัวอย่างฟังชั่นที่สามารถรับข้อมูลเข้าได้ และส่งข้อมูลออกจากฟังชั่นได้ด้วย
ส่งค่าตัวเลขจำนวนเต็มออกจากฟังชั่นได้
รับค่าตัวเลขจำนวนเต็มเข้าฟังชั่นได้
int myFunction(int a){
a = a+100;
return a;
}

จากตัวอย่างทั้งหมดนี้ สรุปได้ว่าชนิดของฟังก์ชันที่เขียนไว้หน้าชื่อนั้นจะเป็นตัวกำหนดว่าฟังก์ชันที่สร้างขึ้นจะใช้รับส่งข้อมูลชนิดใด ตัวอย่างเช่น void หมายถึงไม่ส่งค่าข้อมูลออกจากฟังก์ชัน, int หมายถึงสามารถส่งข้อมูลชนิดเลขจำนวนเต็มออกจากฟังก์ชันได้, float สามารถส่งค่าข้อมูลที่เป็นเลขที่มีจุดทศนิยมได้ หรือ string ก็จะส่งค่าข้อมูลที่เป็นชุดข้อความได้นั่นเอง และการรับข้อมูลเข้าฟังก์ชันก็ต้องทำการประกาศตัวแปรกำหนดชนิดข้อมูลลงในวงเล็บให้สอดคล้องกับข้อมูลที่จะรับเข้ามาหรือส่งออกไป และการประกาศตัวแปรเพื่อจะใช้รับส่งข้อมูลในวงเล็บนั้นสามารถประกาศตัวแปรได้หลายตัวและหลายชนิดได้

โครงสร้างของฟังก์ชันก็มีเท่านี้เอง และนี่คือฟังก์ชันที่ใช้งานจริงใน Arduino

```
void setup(){

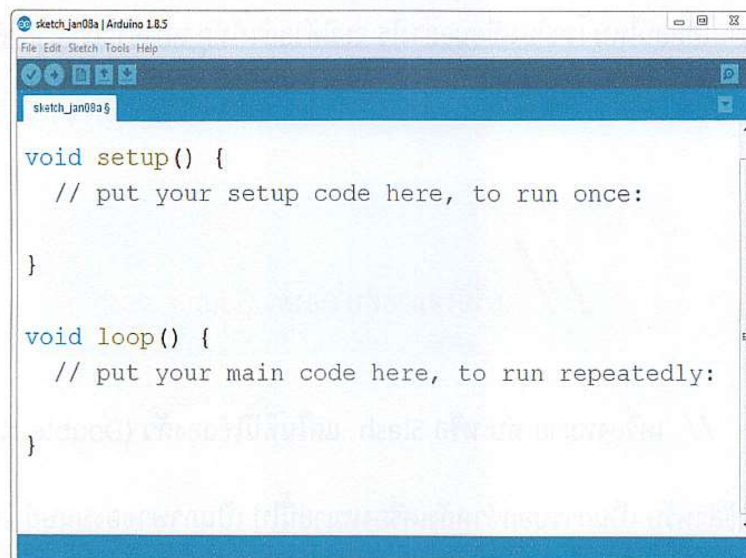
  //ชุดคำสั่งในการตั้งค่าต่าง ๆ

}

void loop(){

  //ชุดคำสั่งที่ต้องการให้ทำงานตลอดเวลา

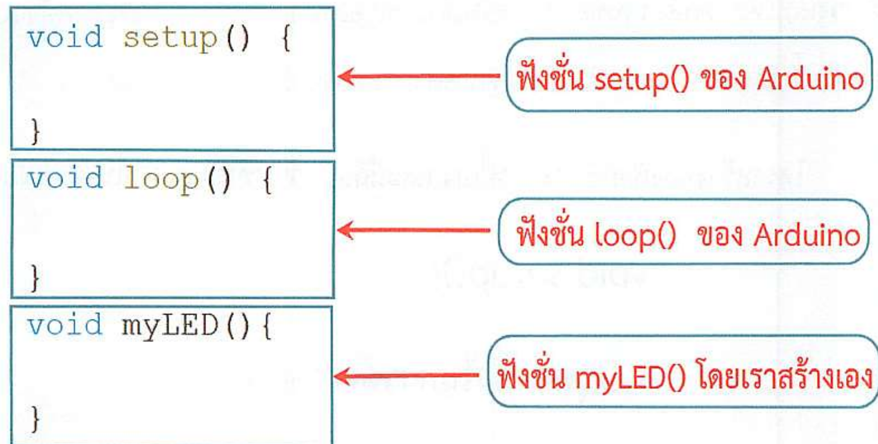
}
```



รูปแสดงตัวอย่างที่ใช้งานจริง

นอกจากฟังก์ชันสำเร็จรูปที่ Arduino สร้างขึ้นมาให้แล้ว เรายังสามารถสร้างฟังก์ชันขึ้นมาใช้งานเองได้อีกด้วย

จากรูปนี้เป็นเพียงตัวอย่างรูปแบบของฟังก์ชันเท่านั้น เป็นฟังก์ชันที่ยังไม่มีการเขียนคำสั่งใดใดลงไป ดังนั้น จากตัวอย่างนี้ โปรแกรมที่เขียนขึ้นถึงแม้จะสามารถอัพโหลดโปรแกรมลงบอร์ดได้แต่บอร์ดจะไม่มีการทำงานใดใดเกิดขึ้นเลย เพราะเรายังไม่ได้เขียนคำสั่งให้ทำงาน



เมื่อเรารู้จักที่มาที่ไปของคำต่างที่ประกอบอยู่ในฟังก์ชัน และ การทำงานของโครงสร้างทั้งหมดของฟังก์ชันแล้ว การเรียนรู้ถึงองค์ประกอบปลีกย่อยเล็กน้อยต่าง ๆ ก็จะไม่เป็นปัญหาอีกต่อไป ในส่วนของฟังก์ชันที่เราสร้างขึ้นมาใช้เองนั้น สำหรับผู้ที่เพิ่งจะเริ่มศึกษาอาจจะยังไม่ค่อยเข้าใจว่าหากเราเขียนฟังก์ชันขึ้นมาใช้งานแล้ว ฟังก์ชันที่เราเขียนขึ้นมาเองนั้นจะทำงานได้อย่างไร ในเมื่อโครงสร้างการทำงานของโปรแกรม ที่เขียนด้วย Arduino นั้นเป็นที่รู้กันอยู่แล้วว่า เมื่อเสร็จจากการทำงานจากการตั้งค่าต่าง ๆ ในฟังก์ชัน setup() แล้ว โปรแกรมก็จะเข้าทำงานใน ฟังก์ชัน loop() แล้วจะวนทำงานอยู่ใน loop() แล้วฟังก์ชันที่เราสร้างจะทำงานได้อย่างไร จะทำงานได้ตอนไหน เราต้องเขียนอย่างไร จะมีตัวอย่างให้ดู หลังจากเรียนรู้คำสั่งทั้งหมดแล้ว

// เครื่องหมายทับ สองตัว Double slash

// เครื่องหมาย ทับ หรือ Slash แต่ในที่นี้ใช้สองตัว (Double slash)

เครื่องหมายนี้ใช้สำหรับ เป็นการบอกว่าหลังเครื่องหมายนี้ไป เป็นภาษาของมนุษย์ ใช้สำหรับเอาไว้เขียนคำอธิบายการทำงานของโปรแกรม เอาไว้ให้คนอ่าน เรียกว่า คอมเมนต์ ข้อความหรือตัวอักษรใดใดหลังเครื่องหมายนี้ จะไม่ถูกนำไปคอมไพล์รวมกับโปรแกรม คือมีไว้ให้คนอ่านเท่านั้น แต่ใช้งานได้เพียงบรรทัดเดียวเท่านั้นหากขึ้นบรรทัดใหม่จะต้องใส่เครื่องหมาย // ใหม่ จึงจะเป็นการคอมเมนต์

void setup() { จุดเริ่มต้นของฟังชั่นนี้

คอมเมนต์

digitalWrite(led,HIGH); //สั่งให้หลอดไฟติด

จุดสิ้นสุดการทำงานของฟังชั่นนี้ }

ภาพแสดงตัวอย่างการใช้งานคอมเมนต์แบบบรรทัดเดียว

/*　　*/

เครื่องหมายทับและดอกจัน

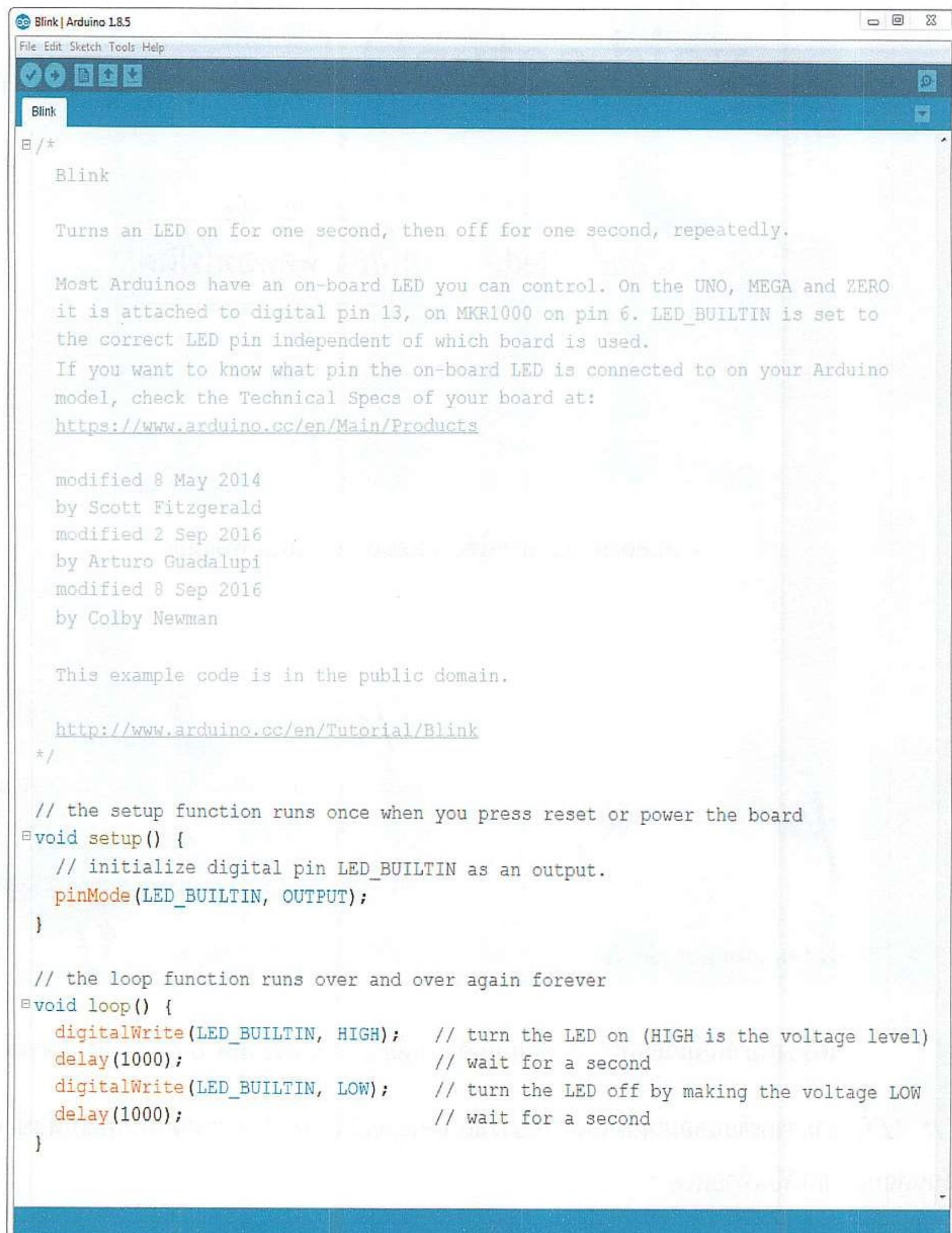
→

/* นี่คือตัวอย่างการใช้งาน
เครื่องหมายที่แสดงให้เห็นถึงการคอมเมนต์
สามารถเขียนได้จำนวนหลายบรรทัด
และจะเขียนกี่บรรทัดก็ได้ เมื่อจบคอมเมนต์
ก็ให้ปิดด้วยเครื่องหมาย */ */

เครื่องหมายนี้มีไว้สำหรับการคอมเมนต์เช่นเดียวกันกับเครื่องหมาย // แต่ต่างกันตรงที่ /* */ นั้นสามารถเขียนคอมเมนต์ได้หลายบรรทัด โดยจุดเริ่มต้นคือ /* จะเขียนกี่บรรทัดก็ได้ เมื่อคอมเมนต์จบแล้วก็ให้ปิดท้ายด้วย */

ตัวคอมไพเลอร์จะไม่นำข้อความในนี้ไปคอมไพล์ คล้ายๆกับว่าเราเขียนไว้ แต่คอมพิวเตอร์จะมองไม่เห็นข้อความในส่วนนี้มีเพียงมนุษย์ที่ตาไม่บอดเท่านั้นที่จะมองเห็นและอ่านมันได้

สังเกตได้จากรูปตัวอย่างต่อไปนี้ เป็นโปรแกรม Blink เป็นตัวอย่างโปรแกรมที่ Arduino มีให้มาอยู่แล้ว จะเห็นว่าคำสั่งโปรแกรมจริงๆนั้นมีอยู่เพียง 5 บรรทัดเท่านั้น ส่วนตัวหนังสือที่เป็นสีเทาจางๆนั้นจะเป็นคอมเมนต์ทั้งหมด ซึ่งเป็นการเขียนอธิบายการใช้งานและการทำงานของโปรแกรม




```
/*
  Blink

  Turns an LED on for one second, then off for one second, repeatedly.

  Most Arduinos have an on-board LED you can control. On the UNO, MEGA and ZERO
  it is attached to digital pin 13, on MKR1000 on pin 6. LED_BUILTIN is set to
  the correct LED pin independent of which board is used.
  If you want to know what pin the on-board LED is connected to on your Arduino
  model, check the Technical Specs of your board at:
  https://www.arduino.cc/en/Main/Products

  modified 8 May 2014
  by Scott Fitzgerald
  modified 2 Sep 2016
  by Arturo Guadalupi
  modified 8 Sep 2016
  by Colby Newman

  This example code is in the public domain.

  http://www.arduino.cc/en/Tutorial/Blink
*/

// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);                       // wait for a second
}
```

ภาพแสดงการใช้งานเครื่องหมาย // และ /* */ ของจริง

Preprocessor Directive พรีโปรเซสเซอร์ ไดเรคทีพ หรือ ตัวประมวลผลก่อน

ตัวประมวลผลก่อน หมายถึงสิ่งที่ตัวคอมไพล์เลอร์จะต้องประมวลผล หรือ อ่านไฟล์ส่วนนี้ก่อนว่าจะให้ทำอะไร มีสิ่งใดที่จะต้องดำเนินการ ก่อนที่จะให้เข้าทำงานในโปรแกรม ในการเขียนนั้นจะเห็นว่ามีเครื่องหมาย # นำหน้า นั่นหมายถึง Preprocessor Directive

ไฟล์ส่วนนี้ ถูกเรียกว่าส่วนหัวของโปรแกรม หรือ เฮดเดอร์ไฟล์ มีอยู่สองชนิดคือ

#include และ #define โดยหลักการคือ จะต้องเขียนไว้ที่ส่วนหัวของโปรแกรม

คือส่วนบรรทัดข้างบนสุดของโปรแกรม โดยทั่วไปจะเขียนไว้ก่อนการประกาศตัวแปร และก่อนฟังก์ชัน แต่หากโปรแกรมของเราไม่ต้องการเรียกใช้เราก็ไม่จำเป็นต้องเขียนแต่อย่างใด

การใช้งาน #include และ #define ใน Arduino นั้นไม่ต่างอะไรกับการใช้งานในภาษา C ทั่วไป จึงสามารถศึกษาถึงรายละเอียดเพิ่มเติมได้จากหนังสือการเขียนโปรแกรมด้วยภาษา C ทั่วไป

#include

#include

#include ใน Arduino นั้นมีไว้สำหรับใช้ดึงไฟล์ไลบรารีต่างๆที่ Arduino สร้างขึ้นมา ให้มาใช้งานได้ในโปรแกรมของเรา หรืออาจจะเป็นไลบรารีที่เราเขียนขึ้นมาเองก็ได้ วิธีใช้งานมีสองแบบคือ สั่งให้ดึงไฟล์มาจากไลบรารี ที่ติดตั้งตามปกติโดยส่วนใหญ่จะอยู่ในโฟลเดอร์มาตรฐานของ Arduino จะใช้เครื่องหมาย < > เพื่อระบุชื่อไฟล์เช่น #include <Wire.h> และ จะใช้เครื่องหมาย “ ” สำหรับดึงไฟล์ไลบรารีที่อยู่ในโฟลเดอร์เดียวกันกับไฟล์โปรแกรม เช่น #include “mylibrary.h” แบบนี้ส่วนใหญ่จะใช้กับไลบรารีที่เราสร้างขึ้นมาใช้งานเอง

ในการเรียกใช้งานไลบรารีนั้น หากไม่ใช่ไลบรารีที่เราเขียนขึ้นมาเอง และ ไม่ใช่ไลบรารีมาตรฐานที่มากับโปรแกรม Arduino IDE หมายความว่าเรายังไม่มีไลบรารีที่ต้องการอยู่ในคอมพิวเตอร์ของเรา จะต้องทำการดาวน์โหลดไลบรารีนั้นมาติดตั้งเสียก่อน จึงจะสามารถเรียกมาใช้งานได้ เรื่องนี้มีปัญหากันมากสำหรับผู้ที่ไม่ศึกษาพื้นฐานให้ดีเสียก่อน หลายๆคนอาจทำตามเว็บไซต์ต่างๆหรือตามคลิปใน youtube แล้วอาจจะคัดลอกโค้ดนั้นมาใช้งาน โดยที่ไม่รู้ความหมายว่าเขาได้เขียนอะไรไว้บ้าง ก็จะเกิดปัญหาพบเจอกับอาการเออเร่อ และ คอมไพล์โปรแกรมไม่ได้ เพราะไม่ทำความเข้าใจในเรื่องของพื้นฐานให้ดีเสียก่อน ส่วนในเรื่องของวิธีการค้นหาและดาวน์โหลดไลบรารีนั้น จะกล่าวถึงในบทต่อๆไป

#include <PWM.h>

ตัวอย่างใช้งานจริง การเรียกใช้ไลบรารี PWM เพื่อจะเขียนโปรแกรมสร้างสัญญาณพัลส์

ไลบรารีตัวอย่างนี้ไม่มีมากับ Arduino IDE หากจะใช้งาน ผู้ใช้จะต้องหาดาวน์โหลดเข้ามาติดตั้งเอง

#include “PWM.h”

ตัวอย่างการใช้งาน หากไฟล์นั้นอยู่ในโฟลเดอร์เดียวกัน กับโปรแกรมที่เราสร้างขึ้น

#define

#define

#define นั้นคือการนิยามหรือการกำหนด คำ หรือ ชื่อ หรือตัวอักษรใดใด ขึ้นมาเพื่อใช้อ้างถึงสิ่งหนึ่ง หากฟังดูเข้าใจยาก และจะพูดง่ายๆให้เป็นภาษาที่ชาวบ้านทั่วไปเข้าใจได้ง่ายๆก็คือ คล้ายๆกับการตั้งชื่อเล่นของคน เช่น เรามีเพื่อนคนหนึ่งชื่อเต็มว่า “นายสมชายใจดีที่สุดในโลก” หากเราจะเรียกชื่อเต็มทุกครั้งไปมันก็จะยาวมากทำให้เสียเวลาในการพูดหรือเขียนชื่อนั้นมาก เราจึงตั้งชื่อเล่นให้เขาว่า “ชาย” คำเดียวสั้นๆ แต่นั่นเป็นเพียงคำอธิบายเพื่อให้เข้าใจถึงความหมายของ #define ได้ง่ายๆเท่านั้น จริงๆแล้ว #define นั้นสามารถทำงานได้หลายอย่าง เช่นการเขียนสูตรคำนวณทางคณิตศาสตร์

ในงานเขียนโปรแกรมสำหรับไมโครคอนโทรลเลอร์ **#define** ถูกนำมาใช้เพื่อกำหนดชื่อต่างๆแทนที่ค่าตัวเลข หรือ ชื่อพอร์ตของไมโครคอนโทรลเลอร์เพื่อความสะดวกในการเขียนโปรแกรม ยกตัวอย่างเช่นเราต้องการจะเขียนโปรแกรมโดยใช้ขา 7 เพื่อควบคุมหลอด LED ให้ติดหรือดับ แต่หากเราใช้เลข 7 อาจทำให้เราลืมหรือสับสนได้ว่าเราต้องการต่อ LED ไว้ที่ขาไหน เราสามารถใช้คำสั่ง #define มากำหนด

#define led 7

ภาพแสดงตัวอย่างการใช้งาน #define

หากประกาศไว้แบบนี้แล้วต่อไปนี้เวลาเราเขียนคำว่า led ลงไปที่ใด โปรแกรมจะเห็นว่านั้นคือเลข 7

หากจะสั่งงานให้ LED ที่ต่ออยู่กับขา 7 ติดขึ้นมา เราจึงเขียนได้ว่า digitalWrite(led,HIGH);

digitalWrite (led,HIGH);

ภาพตัวอย่างการสั่งให้ LED ที่ต่ออยู่กับขา 7 ติด ด้วยการกำหนดค่าด้วย #define

นอกจากนี้แล้ว #define ยังสามารถกำหนดตัวเลขค่าคงที่ เพื่อใช้ในการคำนวณต่างๆ ได้อีกด้วย

โดยข้อดีของการใช้ #define คือจะไม่มีการจองพื้นที่ในหน่วยความจำมากตามขนาดข้อมูลเหมือนตัวแปร เพื่อเก็บค่านั้น จึงทำให้โปรแกรมที่เขียนขึ้นมาขนาดไฟล์ไม่ใหญ่มาก หากเทียบกับการประกาศเป็นตัวแปร

test_define | Arduino 1.8.5

File Edit Sketch Tools Help

test_define §

```
#define t 200
#define led 13
#define led_on digitalWrite(led,HIGH);
#define led_off digitalWrite(led,LOW);
void setup() {
 pinMode(led,OUTPUT);
}
void loop() {
  led_on;
  delay(t);
  led_off;
  delay(t);
}
```

ให้หลอดไฟติด

ให้หลอดไฟดับ

Done uploading.

ตัวอย่างการนิยามคำขึ้นมาใช้แทนคำสั่ง ที่สั่งให้ led ติด และ ดับ ด้วยการใช้ #define

เครื่องหมายและตัวดำเนินการต่าง

ตัวดำเนินการทางคณิตศาสตร์

ตัวดำเนินการทางคณิตศาสตร์ มีอยู่ทั้งหมด 5 ตัว

เครื่องหมาย	ชื่อไทย	ตัวอย่าง	ความหมาย
+	บวก	x = a + b	ให้ x มีค่าเท่ากับ a บวก b
-	ลบ	x = a - b	ให้ x มีค่าเท่ากับ a ลบ b
*	คูณ	x = a * b	ให้ x มีค่าเท่ากับ a คูณ b
/	หาร	x = a / b	ให้ x มีค่าเท่ากับ a หาร b
%	หารเอาเศษ	x = a % b	a หาร b เหลือเศษเท่าใด ให้นำไปใส่ x

ตัวดำเนินการทางคณิตศาสตร์นี้ไม่ได้มีอะไรแปลกใหม่ มันเป็นสิ่งที่เราทุกคนคุ้นเคยกันดีอยู่แล้วนั่นก็คือ เครื่องหมายที่เราใช้กันมาตั้งแต่เรียนประถม คือ บวก ลบ คูณ และหาร ส่วนเครื่องหมาย % นั้นหมายถึงการหารเอาเศษนั่นเอง ซึ่งการนำมาใช้งานก็ยังใช้งานกันแบบธรรมดานี่เองไม่ได้มีอะไรพิสดาร จึงไม่ต้องเป็นกังวลว่ามันจะยาก

```
int a = 50;

int b = 50;

int x = (a + b)-20;
```

จากตัวอย่างนี้จะทำให้ x มีค่าเป็น 80 เพราะเกิดจากการนำค่าของ a และ b มาบวกกัน ผลบวกจะได้ 100 แล้วนำมาลบด้วย 20 ดังนั้น x จึงมีค่า 80 นั่นเอง และ ต้องจำไว้เสมอว่า หากมีการใช้ตัวดำเนินการทางคณิตศาสตร์ที่ซับซ้อน โปรแกรมจะดำเนินการกับข้อมูลที่อยู่ในวงเล็บก่อนเสมอ และหากมีวงเล็บซ้อนกันหลายชั้น โปรแกรมจะเข้าดำเนินการกับข้อมูลที่อยู่วงเล็บชั้นในสุดก่อน ดังนั้นหากเรียงลำดับความสำคัญผิดอาจจะทำให้ผลลัพธ์ที่คำนวณได้นั้นผิดพลาดไปด้วย จึงควรระวังในเรื่องการใช้วงเล็บด้วย

ตัวดำเนินการเปรียบเทียบ

ตัวดำเนินการเปรียบเทียบ

ตัวดำเนินการเปรียบเทียบ มีทั้งหมด 6 ตัว

เครื่องหมาย	ชื่อไทย	วิธีใช้	ความหมาย
==	เท่ากับ	a == b	a เท่ากับ b
!=	ไม่เท่ากับ	a != b	a ไม่เท่ากับ b
<	น้อยกว่า	a < b	a น้อยกว่า b
>	มากกว่า	a > b	a มากกว่า b
<=	น้อยกว่า หรือ เท่ากับ	a <= b	a น้อยกว่าหรือเท่ากับ b
>=	มากกว่า หรือ เท่ากับ	a >= b	a มากกว่าหรือเท่ากับ b

ตัวดำเนินการทางตรรกะ (LOGIC)

ตัวดำเนินการทางตรรกะ

ตัวดำเนินการทางตรรกะ (LOGIC) มีทั้งหมด 3 ตัวเท่านั้น

เครื่องหมาย	ชื่อไทย	วิธีใช้	ความหมาย
&&	and (และ)	a && b	a และ b
||	or (หรือ)	a || b	a หรือ b
!	not (นิเสธ)	อธิบายข้างนอก	อธิบายข้างนอก

ในการใช้ ! หรือ not นั้นเป็นการกลับสถานะทางลอจิก ยกตัวอย่างเช่น

****

int a = 1; //ประกาศตัวแปร a ให้มีค่าเริ่มต้นเป็น 1

a = !a; //แบบนี้ a จะมีค่าเป็น 0 เพราะเป็นการสั่งให้กลับสถานะลอจิก

****

int a = 0; //ประกาศตัวแปร a ให้มีค่าเริ่มต้นเป็น 0

a = !a; //และจากตัวอย่างนี้จะทำให้ a มีค่าเป็น 1

ตัวดำเนินการแบบผสม

เป็นตัวดำเนินการที่ใช้เครื่องหมายหลายๆชนิดมาผสมกันจะได้ความหมายในการดำเนินการขึ้นมาใหม่

เครื่องหมาย	ตัวอย่างการใช้	ความหมาย
++	a++	ให้เพิ่มค่า a ขึ้น 1 ค่า
--	a--	ให้ลดค่า a ลง 1 ค่า
+=	a += b	ให้ a มีค่าเท่ากับ a บวก b
-=	a -= b	ให้ a มีค่าเท่ากับ a ลบ b
*=	a *= b	ให้ a มีค่าเท่ากับ a คูณ b
/=	a /= b	ให้ a มีค่าเท่ากับ a หาร b
%=	a %= b	ให้ a มีค่าเท่ากับเศษที่เหลือจากการหาร b
&=	a &= b	! *อธิบายท้ายตาราง
|=	a |= b	! *อธิบายท้ายตาราง

&= เป็นตัวดำเนินการแบบผสม ที่ใช้สำหรับดำเนินการระดับบิต ส่วนใหญ่จะใช้กับตัวแปรและค่าคงที่ เพื่อบังคับบิตเฉพาะที่อยู่ในตัวแปรให้มีค่าเป็น 0 เป็นการทำงานแบบแอนเกต ส่วนข้อมูลที่จะใช้กับตัวดำเนินการชนิดนี้ได้คือ char ,int, long

```
         0 0 1 1
&
         0 1 0 1
        .........
ผลลัพธ์   0 0 0 1
```

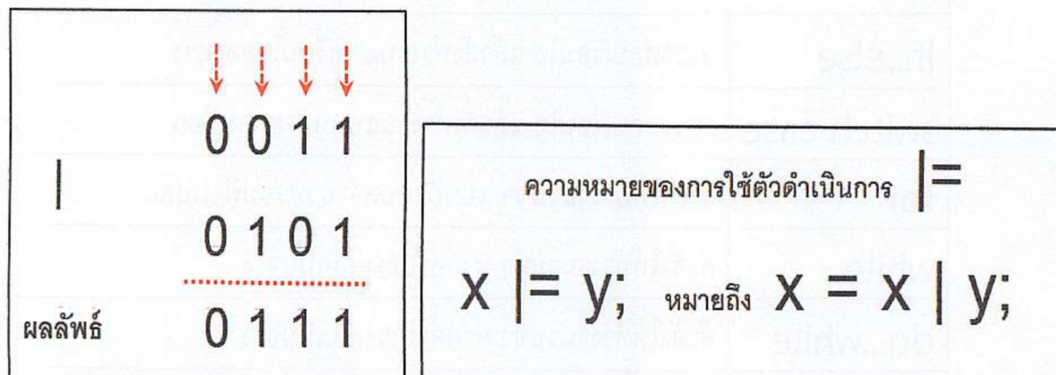
ความหมายของการใช้งานตัวดำเนินการ &=

x &= y; มีความหมายว่า x = x & y;

ตัวอย่างเช่น

```
myByte = B10101010
myByte &= B11111100
ผลลัพธ์ที่ได้  B10101000
```

|= เป็นตัวดำเนินการแบบผสม ที่ใช้สำหรับดำเนินการระดับบิต ส่วนใหญ่จะใช้กับตัวแปรและค่าคงที่ เพื่อบังคับบิตเฉพาะที่อยู่ในตัวแปรให้มีค่าเป็น 1 เป็นการทำงานแบบ or gate ส่วนข้อมูลที่จะใช้กับตัวดำเนินการชนิดนี้ได้คือ char ,int, long



ตัวอย่างเช่น

myByte = B10101010

myByte |= B00000011

ผลลัพธ์ที่ได้ B10101011

Bitwise Operators ตัวดำเนินการระดับบิต

เครื่องหมาย	ความหมาย
&	bitwise and
|	bitwise or
^	bitwise xor
~	เป็นการกลับสถานะบิต
<<	เลื่อนบิตไปทางซ้าย
>>	เลื่อนบิตไปทางขวา

คำสั่งควบคุมการทำงาน

คำสั่งควบคุมการทำงานเป็นคำสั่งที่ใช้ในการเปรียบเทียบค่าของข้อมูลต่างๆและสั่งให้ทำงานตามเงื่อนไขที่กำหนด รวมถึงการสั่งให้ทำงานแบบวนซ้ำที่เรียกว่า loop ใน Arduino นั้นมีอยู่ 10 คำสั่ง

คำสั่งต่างๆเหล่านี้เป็นคำสั่งที่สำคัญมาก ในการเขียนโปแกรมที่ต้องการทำงานที่ซับซ้อนได้ ต้องจำให้ดี

If	ตรวจสอบเงื่อนไข แล้วสั่งทำงานแบบทางเดียว
If...else	ตรวจสอบเงื่อนไข แล้วสั่งทำงานตามเงื่อนไขสองทาง
switch case	ตรวจสอบเงื่อนไข แล้วสั่งทำงานแบบหลายทางเลือก
for	คำสั่งให้ทำงานวนซ้ำ แบบกำหนดจำนวนรอบที่แน่นอน
while	คำสั่งให้ทำงานวนซ้ำ หากค่าในวงเล็บเป็นจริง
do...while	คำสั่งให้ทำงานวนซ้ำ หากค่าในวงเล็บเป็นจริง
break	คำสั่งใช้สำหรับ สั่งให้ออกจาก loop โดยไม่ต้องทำคำสั่งที่เหลือ
continue	คำสั่งให้กลับไปเริ่มทำงานที่ต้น loop โดยไม่ต้องทำคำสั่งที่เหลือ
return	ส่งค่ากลับคืนออกไปจากฟังชั่น
goto	สั่งให้กระโดดไปทำงานใน Label ที่กำหนด * (ไม่ควรใช้บ่อย)

คำสั่งในการควบคุมการทำงานนี้เป็นสิ่งจำเป็นที่ต้องทำความเข้าใจเป็นอย่างมากเพราะเป็นสั่งสำคัญในการควบคุมสั่งงานให้โปรแกรมทำงานตามเงื่อนไขที่เรากำหนดขึ้นมา คำสั่งต่างๆเหล่านี้ จะใช้คู่กับตัวดำเนินการต่างๆ เช่นตัวดำเนินการทางคณิตศาสตร์บวก ตัวดำเนินการเปรียบเทียบ และตัวดำเนินการอื่นๆ แล้วแต่งานที่ทำว่าต้องใช้ตัวดำเนินการใด

ก่อนอื่นต้องทำความเข้าใจก่อนว่าเครื่องหมาย = กับเครื่องหมาย == นั้นมีความหมายไม่เหมือนกัน

เครื่องหมาย = ตัวเดียวนั้นคือการสั่งให้นำ ค่าข้อมูล ที่อยู่ด้าน <u>ขวา</u> ของเครื่องหมาย มาใส่ในตัวแปรที่อยู่ทางด้าน ซ้าย เช่น

a = 100; **เป็นการสั่งให้นำค่า 100 ไปให้ a จะทำให้ a นั้นมีค่าเป็น 100**

a = (200+300); **ตัวอย่างนี้จะให้ a นั้นมีค่าเป็น 500**

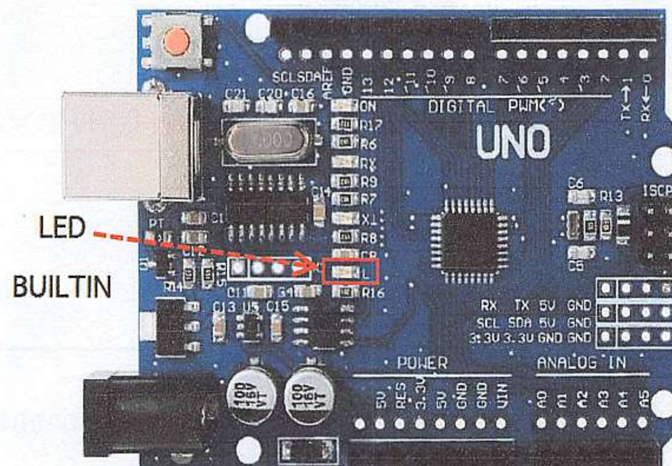
แต่การใช้ == สองตัวนี้เป็นการเปรียบเทียบว่าค่าข้อมูลที่อยู่ทางด้านซ้ายและด้านขวาของเครื่องหมาย == นั้น มีค่าเท่ากันหรือไม่ จะใช้กับคำสั่งควบคุมการทำงานเช่นคำสั่ง if() ใช้เพื่อเปรียบเทียบค่าข้อมูลเพื่อตัดสินใจทำงานบางอย่าง

เช่น if(a == 100) digitalWrite(led,HIGH);

หมายความว่าถ้า a มีค่าเท่ากับ 100 สั่งให้หลอด led ติด

ตัวอย่างโปรแกรมต่อไปนี้หากมีบอร์ดจริง สามารถต่อทดสอบดูการทำงานของจริงได้จะทำให้เข้าใจได้ง่ายขึ้น ให้สังเกตจาก led บนบอร์ดจริงได้เลย โดยไม่ต้องใช้อุปกรณ์มาต่อวงจรเพิ่มเติม

หลอด LED BUILTIN ที่อยู่ในบอร์ด UNO R3 SMD นั้นจะต่ออยู่กับ ขา 13 ตำแหน่งของหลอดที่อยู่ในบอร์ดดูได้ตามรูป หรือหากดูยาก ก็ใช้หลอด LED มาต่อเข้ากับขา 13 ก็ได้




```
#define led 13
int a = 100;
int b = 100;

void setup() {
 pinMode(led,OUTPUT);
 if(a == b) digitalWrite(led,HIGH);
}

void loop() {

}
```

จากตัวอย่างนี้ a กับ b มีค่าเท่ากัน ผลลัพธ์นี้หลอด LED จึงติดนั่นเอง

คำสั่ง if

if เป็นคำสั่งตรวจสอบค่าของตัวแปรหรือข้อมูลที่อยู่ในวงเล็บ ส่วนใหญ่มักใช้กับตัวดำเนินการเปรียบเทียบต่างๆ เช่น == > < != <= >= เพื่อเปรียบเทียบค่าข้อมูลสองตัว หากค่าเป็นจริงก็จะสั่งให้ทำงานตามเงื่อนไขตามคำสั่งที่เราเขียนไว้ หากค่าเป็นเท็จหรือไม่ตรงกับการตรวจสอบที่ต้องการ โปรแกรมก็จะผ่านไปทำงานบรรทัดอื่นต่อไป ตัวอย่างการใช้งาน if แบบทางเดียว

```
#define led 13
int a = 20;
int b = 50;

void setup() {
 pinMode(led,OUTPUT);
 if(a < b) digitalWrite(led,HIGH);
}
void loop() {

}
```

จากรูปจะเห็นว่า a มีค่า 20 และ b มีค่า 50 เมื่อใช้คำสั่ง if ตรวจสอบค่าของทั้งสองตัวในวงเล็บเขียนว่า (a < b) จะเห็นว่าค่านี้เป็นความจริง โปรแกรมจะทำงานตามคำสั่งในวงเล็บปีกกาคือสั่งให้หลอดไฟติด

```
#define led 13
int a = 20;
int b = 50;

void setup() {
 pinMode(led,OUTPUT);
 if(a == b) digitalWrite(led,HIGH);
}
void loop() {

}
```

จากตัวอย่างนี้จะเห็นว่า a กับ b นั้นจริงๆแล้วค่าไม่เท่ากัน แต่เงื่อนไขในการตรวจสอบในวงเล็บได้เปรียบเทียบเอาไว้ว่า ถ้า a กับ b มีค่าเท่ากันจะสั่งให้หลอด LED ติด ผลลัพธ์นี้จึงไม่เข้าเงื่อนไข โปรแกรมจะข้ามคำสั่งนี้ไป และหลอด LED จะไม่ติด

และ if ยังสามารถเขียนซ้อนกันได้หลายชั้น เพื่อตรวจสอบเงื่อนไขย่อยแบบหลายชั้นได้ คำว่าซ้อนกันในที่นี้หมายถึงการเขียนไว้ข้างในวงเล็บปีกกาหรือบล็อกของ if ชั้นนอก และเขียนซ้อนเข้าไปได้หลายชั้น โดยการทำงานนั้นจะเริ่มเข้าทำงานจากเงื่อนไขชั้นนอกสุดเข้าไปทีละชั้น แต่ถ้าหากไม่เข้าเงื่อนไขชั้นนอกสุดแล้วโปรแกรมก็จะไม่เข้าทำงานตรวจสอบเงื่อนไขชั้นใน

If...else

If...else เป็นคำสั่งตรวจสอบเงื่อนไขแบบสองทางเลือก ใช้ในการดำเนินการให้มีทางเป็นไปได้สองทางเลือก หมายความว่า หากเปรียบเทียบด้วย if แล้วไม่เข้าเงื่อนไขก็จะสั่งให้ไปทำงานอีกอย่างหนึ่งใน else และเพื่อให้เห็นผลในการทดลองอย่างชัดเจนเพื่อให้เข้าใจได้ง่ายขึ้นให้ทำการต่อ led เข้ากับขา 7 และ ขา 8 เพื่อดูผลการทดลอง โดยต่อตามตัวอย่าง และเขียนโค้ดตามตัวอย่างต่อไปนี้ แล้วให้ทดลองเปลี่ยนค่าของตัวแปรเพื่อทดลองให้เข้าใจการทำงานได้ยิ่งขึ้น

```
#define led1 7
#define led2 8
int a = 20;
int b = 50;

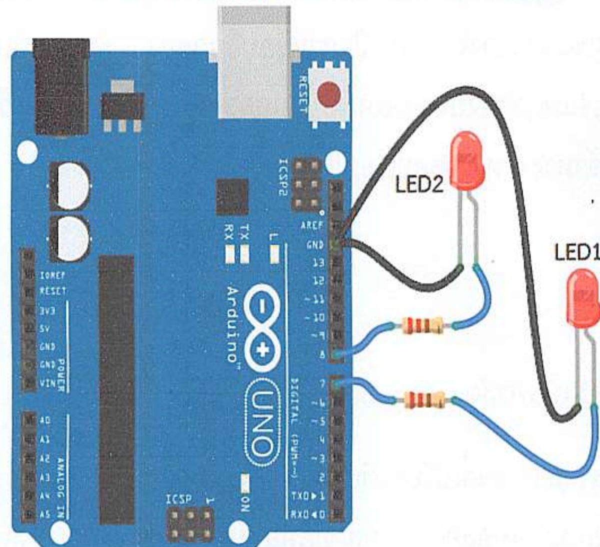
void setup() {
 pinMode(led1,OUTPUT);
 pinMode(led2,OUTPUT);
 if(a == b) digitalWrite(led1,HIGH);
 else digitalWrite(led2,HIGH);
}
void loop() {

}
```

จากตัวอย่างจะเห็นว่าถ้าหาก a เท่ากับ b จะสั่งให้ led1 ที่ต่ออยู่กับขา 7 ติด แต่หากไม่เข้าเงื่อนไขจะเป็นการสั่งให้ led2 ซึ่งต่ออยู่กับขา 8 ติดขึ้นมา นี่คือการใช้งานคำสั่ง if แบบสองทางเลือก รูปแบบของการใช้งาน

ลักษณะนี้จะใช้ if คู่กับ else โดยในวงเล็บของ if นั้นได้ตั้งเงื่อนไขการตรวจสอบเอาไว้ ถ้าหากไม่เข้าเงื่อนไขในวงเล็บ ก็จะให้ไปทำงานในตามคำสั่งของ else ให้ทดลองโดยทำการเปลี่ยนค่าตัวแปร a หรือ b แล้วอัพโหลดโค้ดลงบอร์ดเพื่อดูผลการทดลอง เพื่อให้เข้าใจได้ง่ายยิ่งขึ้น

ภาพประกอบการใช้งาน if แบบสองทางเลือก



if แบบหลายทางเลือก if...else if

if แบบหลายทางเลือก เป็นคำสั่ง if อีกรูปแบบหนึ่งที่ต้องการให้ทำงานได้มากกว่าสองทางเลือกใช้ในการตรวจสอบค่าตัวแปรหลายๆค่าแล้วสั่งให้ทำงานตามเงื่อนไขนั้นๆ และเพื่อให้ง่ายต่อความเข้าใจ เราจะทดลองต่อวงจรทดสอบกันเลย แต่เนื่องจากจะเป็นการทดลองคำสั่ง if แบบหลายทาง เราจึงต้องเปลี่ยนวงจรกันนิดหน่อย โดยการต่อ led เข้าไป 4 ดวงเข้าไปที่ขา 10,11,12,13 ดังรูป

ตัวอย่างการใช้งาน if แบบหลายทางเลือก

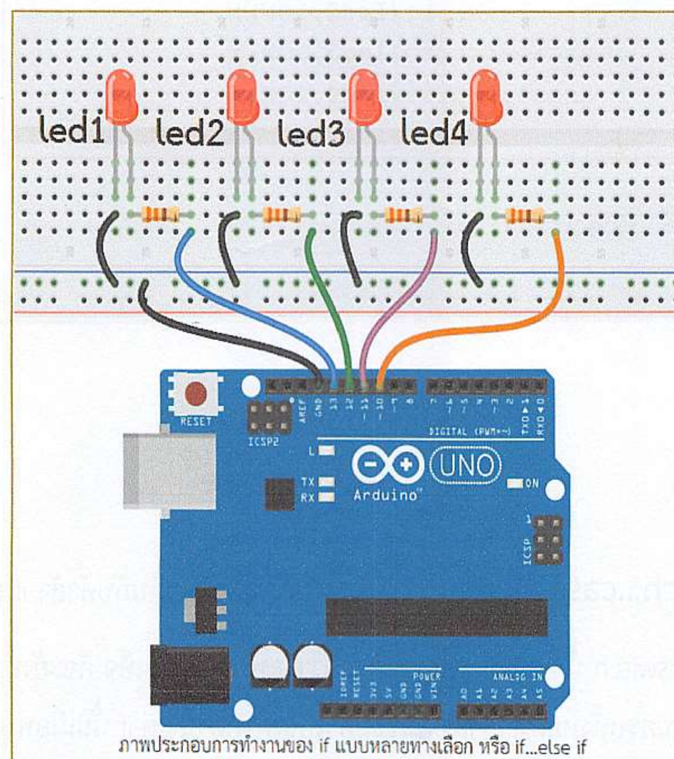
สิ่งที่ต้องจำไว้ให้ดีอีกอย่างหนึ่ง ในการใช้คำสั่ง if และ else คือ หากเป็นการเปรียบเทียบแล้วสั่งให้ทำงานเพียงคำสั่งเดียวนั้น ไม่จำเป็นต้องใส่บล็อกหรือวงเล็บปีกกาก็ได้ แต่หากเปรียบเทียบค่าแล้วต้องการสั่งงานมากกว่าหนึ่งคำสั่ง จะต้องใส่บล็อกหรือวงเล็บปีกกาด้วย ต้องจำให้ดี จากรูปตัวอย่างที่ผ่านมาจะเห็นว่าเป็นการสั่งงานแบบคำสั่งเดียว คือสั่งให้ led เพียงดวงเดียวจึงไม่จำเป็นต้องใส่วงเล็บปีกกาก็ได้ แต่ตัวอย่างต่อไปนี้ เป็นการสั่งหลายคำสั่ง จึงต้องใส่วงเล็บปีกกาด้วย

การตรวจสอบเงื่อนไขแล้วสั่งทำงานเพียงคำสั่งเดียว ไม่ต้องใส่วงเล็บปีกกา ก็ได้

```
if(a == 20)digitalWrite(led,HIGH);
```

ตัวอย่างการสั่งงานหลายคำสั่ง ต้องใส่ { } ด้วย

```
if(a == 10){
  digitalWrite(led1,HIGH);
  digitalWrite(led2,LOW);
  digitalWrite(led3,LOW);
}
else{
  digitalWrite(led1,LOW);
  digitalWrite(led2,HIGH);
  digitalWrite(led3,HIGH);
}
```



ภาพประกอบการทำงานของ if แบบหลายทางเลือก หรือ if...else if

```
#define led1 13
#define led2 12
#define led3 11
#define led4 10

int a = 50;

void setup() {
 pinMode(led1,OUTPUT);
 pinMode(led2,OUTPUT);
 pinMode(led3,OUTPUT);
 pinMode(led4,OUTPUT);
      if(a == 10) digitalWrite(led1,HIGH);
 else if(a == 20) digitalWrite(led2,HIGH);
 else if(a == 30) digitalWrite(led3,HIGH);
 else if(a == 40) digitalWrite(led4,HIGH);
      else{
        digitalWrite(led1,LOW);
        digitalWrite(led2,LOW);
        digitalWrite(led3,LOW);
        digitalWrite(led4,LOW);
      }
}
void loop() {

}
```

ให้ทดลองเปลี่ยนค่าตัวแปร a แล้วอัพโหลดโปรแกรมลงบอร์ดเพื่อทดสอบการทำงาน

switch...case

switch...case เป็นคำสั่งตรวจสอบเงื่อนไขเหมือนกันกับคำสั่ง if แบบหลายทางเลือกแต่จะต่างกันตรงที่คำสั่ง switch นั้นหากตรวจสอบแล้วแม้ว่าจะพบค่าที่เป็นเท็จ ก็จะยังทำการตรวจสอบและทำตามคำสั่งทุกเงื่อนไข จนครบทั้งหมดทุกคำสั่ง แม้จะมีค่าเป็นเท็จก็ตาม แต่ if นั้นเมื่อพบค่าที่เป็นจริงแล้วจะทำงานเพียงคำสั่งเดียวไม่ตรวจสอบค่าที่เหลือ

โดยทั่วไปคำสั่ง switch มักใช้ในการตรวจสอบค่าตัวแปร การใช้งานนั้นจะต้องเขียนตัวแปรที่ต้องการตรวจสอบค่าเอาไว้ในวงเล็บ และคำสั่ง case จะเป็นการเลือกว่า ถ้าหากตัวแปรนั้น มีค่าเท่าใด จะให้สั่งงานให้โปรแกรมทำอะไร

รูปแบบในการเขียน

```
switch (ตัวแปร) {
        case ค่าตัวแปรที่1 : คำสั่งที่ต้องการให้ทำงาน ; break;
        case ค่าตัวแปรที่2 : คำสั่งที่ต้องการให้ทำงาน ; break;
        case ค่าตัวแปรที่3 : คำสั่งที่ต้องการให้ทำงาน ; break;
}
```

ตัวอย่างการใช้งานจริง

```
#define led1 13
#define led2 12
#define led3 11
#define led4 10
int a = 50;
void setup() {
 pinMode(led1,OUTPUT);
 pinMode(led2,OUTPUT);
 pinMode(led3,OUTPUT);
 pinMode(led4,OUTPUT);
  switch(a){
    case 10: digitalWrite(led1,HIGH);
    case 20: digitalWrite(led2,HIGH);
    case 30: digitalWrite(led3,HIGH);
    case 40: digitalWrite(led4,HIGH);
  }
}
void loop() {

}
```

จากตัวอย่างนี้ เราได้กำหนดค่า a ไว้ที่ 50 ดังนั้นจึงไม่เข้าเงื่อนไขใดใด หลอด led จะไม่ติดเลย แต่ถ้าหากค่าของ a เข้าเงื่อนไขได เงื่อนไขหนึ่ง โปรแกรมจะทำงานตามคำสั่งบรรทัดถัดไปทุก case ตัวอย่างเช่น หาก a มีค่า เท่ากับ 10 โปรแกรมก็จะสั่งให้หลอด led ติดหมดทั้ง 4 ดวง เพราะจะลงไปทำตามคำสั่งอีกสามบรรทัดที่เหลือด้วย หรือถ้า a มีค่า 30 โปรแกรมจะทำคำสั่งให้หลอด led3 ติด และจะทำงานตามคำสั่ง case ต่อไปด้วย ถึงทำให้ led4 ติดขึ้นมาด้วย ทั้งๆที่ค่าของ a ไม่ใช่ 40 ดังนั้นหากต้องการให้โปรแกรมทำงานตามคำสั่ง เฉพาะค่าที่เป็นจริงคำสั่งเดียว จะต้องใส่คำสั่ง break; ปิดท้ายทุก case เพราะการใส่ break; ปิดท้ายคำสั่งแต่ละ case นั้นหมายความว่า หากพบเงื่อนไขที่เป็นจริงแล้วให้ออกจากคำสั่ง switch ไป โดยไม่ต้องตรวจสอบ case อื่นๆอีก

```
#define led1 13
#define led2 12
#define led3 11
#define led4 10
int a = 20;
void setup() {
 pinMode(led1,OUTPUT);
 pinMode(led2,OUTPUT);
 pinMode(led3,OUTPUT);
 pinMode(led4,OUTPUT);
  switch(a){
    case 10: digitalWrite(led1,HIGH); break;
    case 20: digitalWrite(led2,HIGH); break;
    case 30: digitalWrite(led3,HIGH); break;
    case 40: digitalWrite(led4,HIGH); break;
  }
}
void loop() {

}
```

ให้ทดลองเปลี่ยนค่าของ a แล้วอัพโหลดโปรแกรม เพื่อดูผลการทดลอง จะให้เข้าใจยิ่งขึ้น

หากปิดท้ายด้วย break; เมื่อ a มีค่า 30 โปรแกรมจะสั่งให้ led3 ติดเพียงดวงเดียวเท่านั้น

และโปรแกรมจะออกจากคำสั่ง switch ไปโดยไม่ตรวจสอบ case อื่นๆที่เหลือ

การใช้งาน switch...case นั้น หากพบเงื่อนไข case ที่ต้องการแล้ว แต่ต้องการเขียนคำสั่งให้ทำงานเกิน 1 คำสั่ง จะต้องใส่วงเล็บปีกกาที่ case { } ด้วย แต่หากเป็นการสั่งงานคำสั่งเดียว ไม่ต้องใส่วงเล็บปีกกาก็ได้

ตัวอย่างการใส่วงเล็บปีกกา เพื่อสั่งงานหลายคำสั่ง

```
switch(a){
  case 10:{
    digitalWrite(led1,HIGH);
    digitalWrite(led2,LOW);
  }break;
  case 20:{
    digitalWrite(led1,LOW);
    digitalWrite(led2,HIGH);
  }break;
}
```

****

```
#define led1 13
#define led2 12
#define led3 11
#define led4 10
int a = 30;
void setup() {
 pinMode(led1,OUTPUT);
 pinMode(led2,OUTPUT);
 pinMode(led3,OUTPUT);
 pinMode(led4,OUTPUT);
  switch(a){
    case 10: digitalWrite(led1,HIGH);break;
    case 20: digitalWrite(led2,HIGH); break;
    case 30: {
                a = 40;
                digitalWrite(led3,HIGH);
             } break;
    case 40: digitalWrite(led4,HIGH); break;
  }
}
void loop() {

}
```

for

for เป็นคำสั่งให้ทำงานแบบวนซ้ำ ที่เรียกว่าลูป เป็นการทำงานวนซ้ำ แบบมีจำนวนครั้งที่กำหนดได้และรู้การทำงานที่แน่นอนว่าต้องการวนซ้ำกี่รอบ ส่วนใหญ่มักใช้เพื่อให้นับค่าตัวแปรแบบมีจุดมุ่งหมายปลายทางที่แน่นอน การใช้งานลูปนี้จะต้องเขียนตามเงื่อนไขที่ถูกต้องเท่านั้น โดยต้องกำหนดค่าเริ่มต้นก่อนตามด้วยนิพจน์เงื่อนไข และปรับค่าตัวนับว่าจะให้เพิ่มขึ้นหรือลดค่าลงก็ได้

```
for(ค่าเริ่มต้น; เงื่อนไข; ตัวนับเพิ่มหรือลด)
```

```
for(a = 0; a < 100; a++)
```

จากตัวอย่างนี้จะวนลูปการนับ 100 ครั้ง โดยเริ่มจาก 0 และเพิ่มค่าขึ้นครั้งละ 1 ค่าจนครบ 100 รอบ โดยการนับแต่ละรอบจะเข้าทำงานตามคำสั่งในวงเล็บปีกกา 1 ครั้ง นับ 100 รอบก็เข้าทำงาน 100 ครั้ง หรือถ้าตั้งเงื่อนไขไว้แค่ 10 รอบ โปรแกรมก็จะวนทำงานซ้ำๆจำนวน 10 รอบ และเพื่อความง่ายต่อการเข้าใจ ให้ทำการต่อหลอด LED เข้าที่ขา 8 และทดลองเขียนโค้ดโปรแกรมตามตัวอย่าง แล้วสังเกตการณ์ติดดับของหลอด LED จะทำให้เข้าใจการทำงานได้ง่ายขึ้น

```
for(int a = 0; a < 100; a++) {
   //คำสั่งที่ต้องการให้ทำงานในแต่ละรอบของการนับ
}
```

ให้ทดลองเขียนโปรแกรมการวนทำงานซ้ำด้วย for ตามตัวอย่าง แล้วกดดูผลที่ Serial monitor

```
void setup() {
  Serial.begin(9600);            //กำหนดความเร็วในการรับส่งข้อมูล 9600 บิต ต่อวินาที
}

void loop() {
 for(int a=0; a<100; a++) { //วนลูพ for เพิ่มค่า a ขึ้นทีละ 1 ค่า จำนวน 100 รอบ โดยเริ่มจาก 0
     Serial.println(a);          //ส่งข้อมูลผ่าน Serial port เพื่อแสดงผลที่ Serial monitor
     delay(1000);                //หน่วงเวลาไว้ 1 วินาที
 }
}
```

ตัวอย่างการวนทำงานซ้ำด้วย for แสดงผลด้วยหลอด LED จะเห็นผลการทำงานย่างชัดเจน

```
//-----------------------------------------------------------------
#define led 8                      //ประกาศชื่อแทนขา 8 ว่า led
//-----------------------------------------------------------------
void setup() {                     //เริ่มฟังชั่น setup()
  pinMode(led,OUTPUT);             //ประกาศให้ขา 8 ทำงานเป็น เอาต์พุต
  for(int i=0;i<5;i++){            //สั่งให้วนทำงานซ้ำ 5 รอบ ด้วย for
    digitalWrite(led,HIGH);        //สั่งให้ LED ติด
    delay(500);                    //หน่วงเวลา 500ms (ครึ่งวินาที)
    digitalWrite(led,LOW);         //สั่งให้ LED ดับ
    delay(500);                    //หน่วงเวลา 500ms (ครึ่งวินาที)
  }
  delay(3000);                     //หน่วงเวลา 3000ms (3 วินาที)
}                                  //สิ้นสุดฟังชั่น setup()
//-----------------------------------------------------------------
void loop() {                      //เริ่มฟังชั่น loop()
  for(int i=0;i<10;i++){           //สั่งให้วนทำงานซ้ำ 10 รอบ ด้วย for
    digitalWrite(led,HIGH);        //สั่งให้ LED ติด
    delay(200);                    //หน่วงเวลา 200ms (0.2 วินาที)
    digitalWrite(led,LOW);         //สั่งให้ LED ดับ
    delay(200);                    //หน่วงเวลา 200ms (0.2 วินาที)
  }
  delay(2000);                     //หน่วงเวลา 2000ms (2 วินาที)
}                                  //สิ้นสุดฟังชั่น loop()
//-----------------------------------------------------------------
```

อธิบายความหมายสิ่งที่อยู่ในวงเล็บ for

```
for( int i = 0; i < 5; i++ )
```

- `int i = 0;` — ประกาศตัวแปรชนิดเลขจำนวนเต็ม ชื่อว่า i และกำหนดค่าให้มีค่าเริ่มต้นที่ 0
- `i < 5;` — ถ้าหากค่าของ i น้อยกว่า 5
- `i++` — ให้เพิ่มค่า i ขึ้น 1 ค่า

และนอกจากลูป for จะทำงานแบบวนซ้ำแบบนับจำนวนรอบได้แล้ว ยังสามารถใช้งานให้ทำงานแบบวนลูปไม่รู้จบได้อีกด้วย โดยการใส่เครื่องหมาย ;; ลงในวงเล็บ เพียงเท่านี้โปรแกรมก็จะทำงานวนอยู่ในลูป จากตัวอย่างด้านล่างนี้เมื่อเริ่มทำงาน led จะติด และจะติดอยู่ตลอดไป ถึงแม้ว่าในฟังก์ชั่น loop จะเขียนสั่งงานให้หลอดดับ แต่การใช้ for(;;) จะทำให้โปรแกรมทำงานวน ติดอยู่ในลูปนี้ ตลอดไป จึงไม่สามารถเข้าทำงานในฟังก์ชัน void loop() ได้

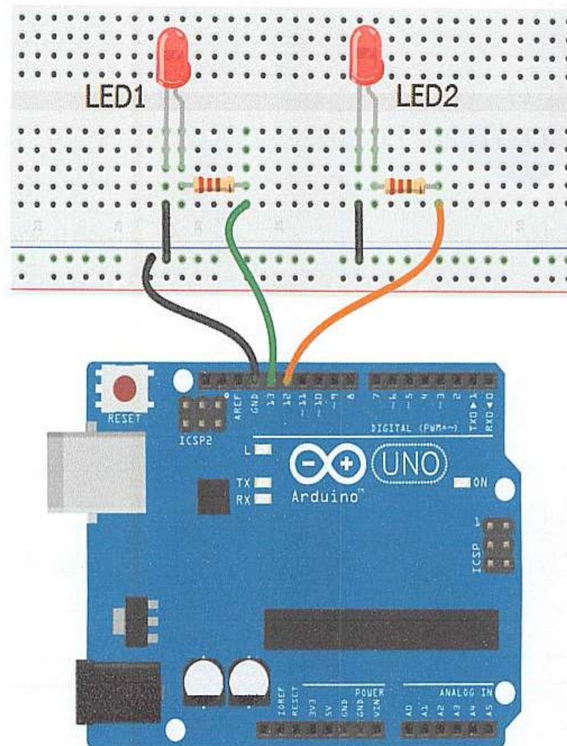
```
//-----------------------------------------------------------
#define led 8                    //กำหนดให้ขา 8 ชื่อว่า led
//-----------------------------------------------------------
void setup() {
  pinMode(led,OUTPUT);           //ให้ขา 8 ทำงานเป็น เอาต์พุต

  for(;;){                       //ใส่เครื่องหมาย ;; ลงในวงเล็บ
    digitalWrite(led,HIGH);      //ให้หลอดติด
  }

}
//-----------------------------------------------------------
void loop() {
  digitalWrite(led,LOW);         //ให้หลอดดับ
}
//-----------------------------------------------------------
```

และลูป for ยังสามารถเขียนซ้อนลูปได้อีกด้วย โดยการทำงานนั้น จะเข้าทำงานลูปชั้นในตามจำนวนครั้งจากค่าตัวแปรของลูปนอก หากดูจากตัวอย่างต่อไปนี้จะเห็นว่าลูปชั้นนอกให้นับ 5 รอบ และลูปชั้นในให้นับ 3 รอบ โดยกำหนดให้ led1 กระพริบแสดงการนับของลูปชั้นใน และ led2 กระพริบแสดงการนับของลูปชั้นนอก ให้ทดลองอัพโหลดโค้ดโปรแกรมนี้ดู จะเห็นว่าเมื่อโปรแกรมทำงาน led1 จะกระพริบ 3 ครั้ง และ led2 จะกระพริบ 1 ครั้ง และจะวนทำงานอย่างนี้ 5 รอบตามการนับของลูปชั้นนอกนั่นเอง

วงจรที่ใช้สำหรับทดลองการทำงานของลูป for แบบซ้อนลูป




```
#define led1 13
#define led2 12
void setup() {
  pinMode(led1,OUTPUT);
  pinMode(led2,OUTPUT);
  for(int x=0;x<5;x++){
      for(int y=0;y<3;y++){
       digitalWrite(led1,HIGH);
       delay(500);
       digitalWrite(led1,LOW);
       delay(500);
      }
   digitalWrite(led2,HIGH);
   delay(500);
   digitalWrite(led2,LOW);
   delay(500);
  }
}

void loop() {

}
```

ลูพ for ชั้นใน LED1 กระพริบ 3 ครั้ง

ลูพ for ชั้นนอก LED2 กระพริบ 5 ครั้ง

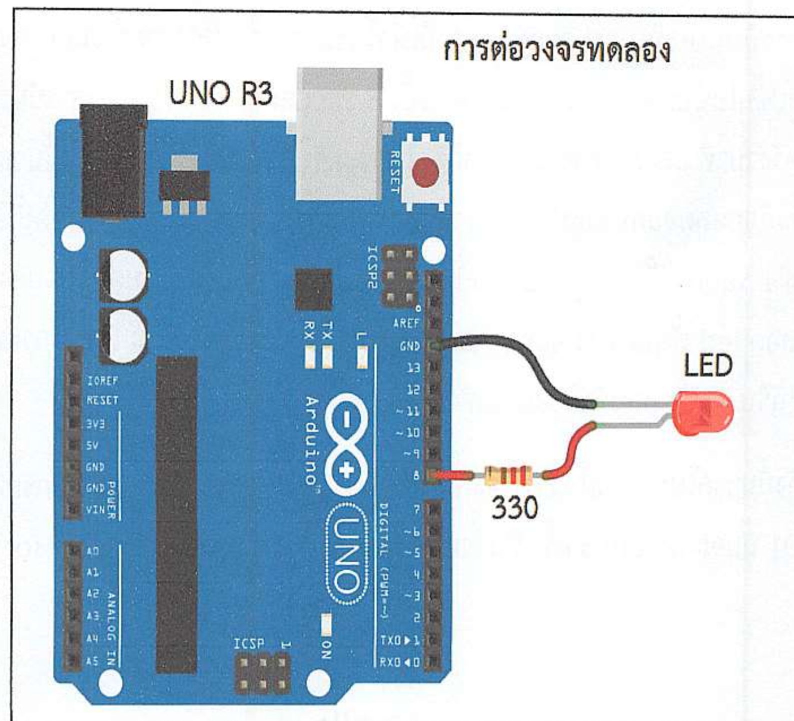
การนับแบบซ้อนลูป จากตัวอย่างนี้เมื่อโปรแกรมเข้าสู่ลูป for ชั้นนอก ก็จะพบกับลูปชั้นในอีกชั้นหนึ่ง โดยลูปชั้นในนั้นเขียนสั่งงานให้วนทำงานซ้ำ 3 รอบและในแต่ละรอบของการนับนั้น เขียนสั่งงานเอาไว้ให้ led1 ติดครึ่งวินาที และดับ ครึ่งวินาที ซึ่งก็คือไฟกระพริบนั่นเอง ดังนั้นลูปชั้นในนั้นจะสั่งให้ led1 กระพริบ 3 ครั้ง จากนั้นก็จะหลุดออกจากลูปชั้นใน มาเจอคำสั่งให้ LED2 ติด และดับ ทั้งหมดนี้เป็นการทำงานของลูปชั้นนอก เพียง 1 รอบเท่านั้น แต่ลูปชั้นนอกนั้นถูกเขียนสั่งให้ทำงานซ้ำทั้งหมด 5 รอบดังนั้นให้สังเกตจากการทำงานของหลอด led ทั้งสองดวง จะพบว่าเมื่อโปรแกรมเริ่มทำงานจะทำให้ LED1 กระพริบ 3 ครั้งก่อน แล้ว LED2 จะกระพริบ 1 ครั้ง สลับกันไปอย่างนี้ทั้งหมด 5 รอบ

ดังนั้นจากโปรแกรมนี้ LED1 จะกระพริบทั้งหมด 15 ครั้ง และ LED2 จะกระพริบทั้งหมด 5 ครั้ง เพราะว่า LED1 นั้นจะกระพริบ 3 ครั้งเป็นจำนวนทั้งหมด 5 รอบ ตามการนับของลูป for ชั้นนอก

while

while เป็นการทำงานแบบวนซ้ำคล้ายกับ for แต่ต่างกันตรงที่ while นั้นจะทำการตรวจสอบค่าในวงเล็บก่อน โดยการตรวจสอบเงื่อนไขนี้เป็นไปได้สองทาง คือจริงหรือเท็จเท่านั้น หากเงื่อนไขในวงเล็บมีค่าเป็นจริง จึงจะเข้าทำงานตามคำสั่งในวงเล็บปีกกา แต่หากเป็นเท็จโปรแกรมจะไม่เข้าทำงานตามคำสั่งในลูปนี้ แต่จะข้ามไปเลย คำสั่ง while นี้เราสามารถให้โปรแกรมทำงานวนอยู่ในลูปนี้ได้ด้วยการใส่ค่าในวงเล็บให้เป็นจริง โปรแกรมก็จะทำงานอยู่ในลูปนี้ตลอดกาล เพียงแค่ใส่เลข 1 ลงไปในวงเล็บเท่านั้นเองเพราะตัวเลข 1 นั้นในทางตรรกะถือว่ามีค่าเป็นจริง และ 0 มีค่าเป็นเท็จ ตัวอย่างต่อไปนี้จะเห็นได้ว่าแม้ว่าจะเขียนสั่งงานไว้ในฟังก์ชัน loop ว่าให้หลอด led ดับ แต่ความเป็นจริงแล้วหลอดจะติดกระพริบตลอดเวลา เพราะในฟังก์ชัน setup() นั้นได้เขียนลูป while() และใส่ค่าให้เป็นจริงเข้าไว้โดยใส่เลข 1 ลงไป ดังนั้นเมื่อโปรแกรมทำงานมาถึงลูปนี้แล้วพบค่าเป็นจริง โปรแกรมจะติดการทำงานอยู่ใน while() และไม่ออกไปทำงานในที่อื่นอีกเลย เพียงเพราะค่าในวงเล็บนั้นมีค่าเป็นจริง หากต้องการให้โปรแกรมออกจากลูป while() นี้ต้องทำให้เงื่อนไขในวงเล็บนั้นมีค่าเป็นเท็จก่อน โปรแกรมจึงจะออกไปทำงานอื่นๆได้

สรุปได้ว่าการทำงานของ **while(1)** และ **for(;;)** นั้นมีผลเหมือนกันทุกประการ




```
#define led 8                    //กำหนดชื่อ led แทนขา 8

void setup() {                   //เริ่มฟังชั่น setup
  pinMode(led,OUTPUT);           //ตั้งค่าให้ขา 8 ทำงานเป็นเอาต์พุต
  while(1){                      //เริ่ม loop while และกำหนดค่าให้เป็นจริง
    digitalWrite(led,HIGH);      //สั่งให้ led ติด
    delay(200);                  //หน่วงเวลา 0.2 วินาที
    digitalWrite(led,LOW);       //สั่งให้ led ดับ
    delay(200);                  //หน่วงเวลา 0.2 วินาที
  }                              //จบลูพ while
}                                //จบฟังชั่น setup

void loop() {                    //เริ่มฟังชั่น loop
   digitalWrite(led,LOW);        //สั่งให้หลอด led ดับ
}                                //จบฟังชั่น loop
```

นอกจากการกำหนดค่าในวงเล็บได้โดยตรงแล้ว ลูป while ยังสามารถใช้ในการตรวจสอบค่าตัวแปร ได้เช่นเดียวกับคำสั่ง if แต่ที่ต่างกันคือ ลูป while นั้นจะวนทำงานอยู่ในลูปไม่ออกไปทำงานที่อื่นถ้าหากค่าใน วงเล็บยังคงเป็นจริง มันจะวนทำงานอยู่ในนั้นอย่างนั้นจนกว่าค่าในวงเล็บจะเป็นเท็จ แต่คำสั่ง if นั้นจะเข้า ทำงานตามคำสั่งเมื่อค่าในวงเล็บเป็นจริงเช่นกัน แต่เมื่อทำงานเสร็จแล้วก็จะออกไปทำงานตามคำสั่งถัดไป ตามปกติ ดังนั้นเมื่อเข้าใจดีแล้วเวลาจะนำคำสั่งใดไปใช้ ต้องเลือกให้เหมาะสมกับงานที่จะทำ

ตัวอย่างต่อไปนี้เป็นการทดลองใช้ while เพื่อตรวจสอบค่าตัวแปรสองตัวในวงเล็บ ถ้าหากค่าเป็นจริงโปรแกรมจะเข้าทำงานในลูป while และวนทำงานอยู่ในนั้น แต่เมื่อใดที่ค่าในวงเล็บเป็นเท็จ โปรแกรมก็จะเข้าทำงานในฟังก์ชัน loop() ตามปกติ ให้ทดลองปรับเปลี่ยนค่าตัวแปรและดูผลที่เกิดขึ้น จะทำให้เข้าใจง่ายขึ้น จากตัวอย่างนี้ a มีค่า 100 และ b มีค่า 200 และในวงเล็บ while เขียนไว้ว่า while(a<b) หมายความว่า ค่าของ a มีค่าน้อยกว่าค่าของ b ซึ่งเป็นความจริง ดังนั้น จากผลการทดลองนี้ เราจะเห็นว่าหลอด led นั้นติดกระพริบด้วยความเร็ว 200 มิลลิวินาทีตามค่าการหน่วงเวลาด้วยคำสั่ง delay(200);

```
#define led 8                     //กำหนดชื่อ led แทนขา 8
int a = 100;             //สร้างตัวแปรชนิดเลขจำนวนเต็มชื่อว่า a และกำหนดค่าเป็น 100
int b = 200;             //สร้างตัวแปรชนิดเลขจำนวนเต็มชื่อว่า b และกำหนดค่าเป็น 200
void setup() {                    //เริ่มฟังชั่น setup
  pinMode(led,OUTPUT);            //ตั้งค่าให้ขา 8 ทำงานเป็นเอาต์พุต
  while(a < b){                   //เข้าทำงานในลูพ while ถ้าหากค่า a น้อยกว่า b
    digitalWrite(led,HIGH);       //สั่งให้ led ติด
    delay(200);                   //หน่วงเวลา 0.2 วินาที
    digitalWrite(led,LOW);        //สั่งให้ led ดับ
    delay(200);                   //หน่วงเวลา 0.2 วินาที
  }                               //จบลูพ while
}                                 //จบฟังชั่น setup

void loop() {                     //เริ่มฟังชั่น loop
   digitalWrite(led,HIGH);        //สั่งให้หลอด led ติดค้าง
}                                 //จบฟังชั่น loop
```

do...while

คำสั่ง do...while นั้นเป็นคำสั่งที่ทำงานคล้ายๆกันกับ while แต่ต่างกันที่โปรแกรมจะเข้าไปทำงานตามคำสั่งที่เขียนไว้เข้าในลูปก่อน 1 รอบแล้วจึงออกมาตรวจสอบค่าในวงเล็บทีหลังว่าค่านั้นเป็นจริงหรือเท็จ หากเป็นเท็จก็จะไม่เข้าไปทำงานอีก แต่หากค่าเป็นจริงก็จะวนกลับเข้าไปทำงานใหม่ และวนทำงานอยู่อย่างนี้จนกว่าค่าในวงเล็บจะเป็นเท็จ สิ่งที่แตกต่างกันกับ while ก็คือ do...while นั้น ไม่ว่าค่าในวงเล็บจะเป็นจริงหรือเป็นเท็จ โปรแกรมก็จะเข้าทำงานตามคำสั่งในลูปก่อน 1 รอบ แล้วจึงออกมาดูค่าในวงเล็บว่าเป็นจริงหรือเท็จ หากค่าเป็นจริงก็จะเข้าไปทำงานใหม่ และหากค่าเป็นเท็จก็จะไม่กลับเข้าไปทำงานอีก ส่วนลูป while นั้นจะทำการตรวจสอบค่าในวงเล็บก่อนเสมอหากค่าเป็นเท็จจะไม่เข้าทำงานเลย จะเข้าทำงานก็ต่อเมื่อค่าในวงเล็บเป็นจริงเท่านั้น สรุปว่า while นั้นจะตรวจสอบก่อนถ้าเป็นจริงจึงจะเข้าทำงาน แต่ do...while นั้นจะเข้าทำงานก่อน แล้วตรวจสอบทีหลัง มันจะเข้าทำงานก่อน 1 รอบโดยไม่สนใจว่าค่าในวงเล็บเป็นจริงหรือเท็จ

เพราะเขียน do ไว้ก่อน และหากแปลกันตามความหมายของ do แล้วก็หมายถึงการสั่งให้ทำงานตามคำสั่งที่เขียนไว้ในวงเล็บปีกกาหลังคำว่า do ก่อนแล้วค่อยเช็ค while() ทีหลัง

ตัวอย่างนี้ถ้าหาก a น้อยกว่า 100 จะสั่งให้เพิ่มค่าขึ้นเรื่อยๆจนกว่า a จะไม่น้อยกว่า 100

```
do
{
 a = a++;
}while(a < 100);
```

do...while โปรแกรมจะเข้าทำคำสั่งในปีกกาก่อนหนึ่งรอบ จึงจะออกมาตรวจค่าในวงเล็บ while(a < 100) ทีหลังในการใช้งาน do...while นั้นต้องเลือกใช้งานให้เหมาะกับงานเพราะหากใช้งานผิดประเภทแล้วโปรแกรมอาจทำงานผิดพลาดได้

โปรแกรมทดลองใช้งานจริง โดยต่อ LED ไว้ที่ขา 8

```
#define led 8                       //กำหนดชื่อ led แทนขา 8
int a = 100;                        //สร้างตัวแปร a ให้มีค่า 100
int b = 200;                        //สร้างตัวแปร b ให้มีค่า 200

void setup() {
  pinMode(led,OUTPUT);              //กำหนดให้ขา 8 ทำงานเป็นเอาต์พุต
  do{                               // จุดเริ่มต้นคำสั่ง do
    digitalWrite(led,HIGH);         //สั่งให้หลอด LED ติด
    delay(3000);                    //หน่วงเวลา 3 วินาที
    digitalWrite(led,LOW);          //สั่งให้หลอด LED ดับ
  }                                 // จุดสิ้นสุดคำสั่ง do
  while(a > b);                     //คำสั่ง while
}

void loop() {

}
```

จากโปรแกรมตัวอย่างในรูปเป็นการใช้คำสั่ง do...while สั่งให้ตรวจสอบค่าของตัวแปร a กับ b การทำงานของโปรแกรมนั้นเมื่อเริ่มทำงาน LED จะติดค้างเป็นเวลา 3 วินาที เนื่องจากในปีกกาของ do

นั้นได้เขียนสั่งงานเอาไว้ว่าให้หลอด LED ติด 3 วินาที แล้วดับ เสร็จแล้วจึงจะออกมาตรวจสอบค่าใน while(a > b); ทีหลัง หากค่าเป็นเท็จก็จะออกจากลูปนี้ไปทำตามคำสั่งถัดไป แต่ถ้าหากค่าเป็นจริง จึงจะกลับเข้าไปทำงานที่ do ใหม่ จากโปรแกรมนี้ a = 100 และ b = 200 เมื่อเขียนว่า

while(a > b); หมายความว่าให้เข้าลูปนี้ เมื่อ a มากกว่า b

แต่ความจริงคือ a นั้นมีค่าน้อยกว่า b เพราะ a มีค่า 100 เท่านั้น แต่ b นั้นมีค่าถึง 200

ดังนั้นผลลัพธ์ที่ได้จึงเป็นเท็จ โปรแกรมจะไม่กลับเข้าไปทำงานในลูปอีก แต่ก็ได้ทำงานในลูปไปแล้ว 1 รอบ นี่เป็นการทดลองการทำงานของ do...while ที่เห็นผลได้ง่ายๆ และสรุปได้ว่า โปรแกรมจะเข้าทำงานในลูปก่อน 1 รอบ แล้วจึงออกมาตรวจสอบเงื่อนไขทีหลัง จึงต่างจากการทำงานของ while เฉยๆ เพราะ while() นั้นจะเข้าทำงานในลูปก็ต่อเมื่อค่าในวงเล็บเป็นจริงเท่านั้น

เพื่อความเข้าใจอันดี ให้ผู้อ่านทำการทดลองเปลี่ยนค่าตัวแปร a กับ b ให้ a มีค่ามากกว่า b แล้วทำการทดลองอัพโหลดโปรแกรมดูผลการทำงานที่เกิดขึ้นจะทำให้เข้าใจดีขึ้น

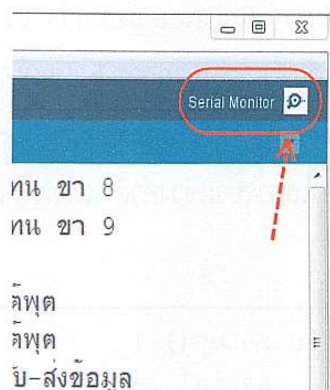
คำสั่ง goto

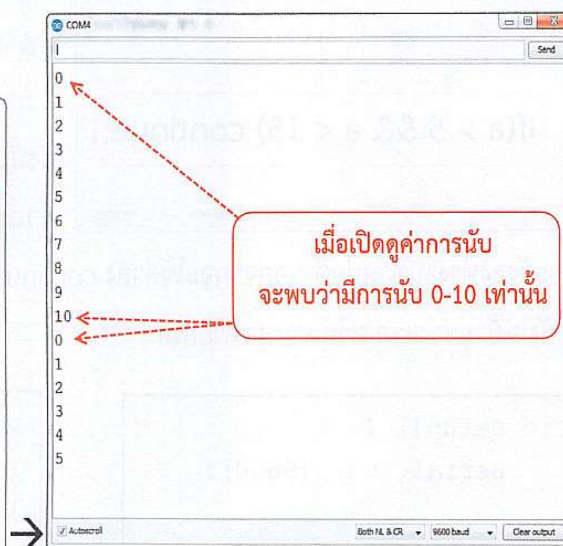
goto เป็นคำสั่งที่ใช้สำหรับการกระโดดการทำงานไปยัง LABEL ที่กำหนด โดยจะข้ามคำสั่งทุกคำสั่ง และไปทำงานในบรรทัดที่กำหนดและบรรทัดต่อๆไปเลย การใช้งานคำสั่ง goto นั้นจะใช้คู่กับ LABEL ในการประกาศ LABEL นั้นทำได้โดยการตั้งชื่อขึ้นมาแล้วตามด้วยเครื่องหมาย : ส่วนการตั้งชื่อขึ้นมาใช้งานนั้นก็ให้ใช้ชื่อให้สอดคล้องกับการทำงานที่จุดนั้นเพื่อป้องกันการสับสน จากโปรแกรมตัวอย่างตั้งชื่อ LABEL ว่า led2_ON : เป็นการบอกให้รู้ว่าการทำงานจุดนี้คือสั่งให้ led2 ติดนั่นเอง ในการทดลองโปรแกรมนี้ให้ทำการต่อหลอด led1 ไว้ที่ขา 8 และต่อ led2 ไว้ที่ขา 9 และทำการทดลองดูผลที่เกิดขึ้น

```
#define led1 8                              //ตั้งชื่อ led1 ขึ้นมาใช้แทน ขา 8
#define led2 9                              //ตั้งชื่อ led2 ขึ้นมาใช้แทน ขา 9
void setup() {
  pinMode(led1,OUTPUT);                     //ให้ขา 8 ทำงานเป็นเอาต์พุต
  pinMode(led2,OUTPUT);                     //ให้ขา 9 ทำงานเป็นเอาต์พุ
}
void loop() {
    for(int i=0;i<50;i++){                  //วนลูพนับเพิ่มค่า i จำนวน 50 รอบ
     if(i == 10) goto led2_ON;              //ถ้า i เท่ากับ 10 ให้กระโดดไปที่ led2_ON
        digitalWrite(led1,HIGH);            //ให้ led1 ติด
        delay(200);                         //หน่วงเวลา 200 มิลลิวินาที
        digitalWrite(led1,LOW);             //ให้ led1 ดับ
        delay(200);                         //หน่วงเวลา 200 มิลลิวินาที  <---- ข้าม
    }
        digitalWrite(led1,HIGH);            //ให้ led1 ติด
        delay(5000);                        //หน่วงเวลา 5 วินาที
  led2_ON: digitalWrite(led2,HIGH);         //ให้ led2 ติด
           delay(500);                      //หน่วงเวลาครึ่งวินาที
           digitalWrite(led2,LOW);          //ให้ led2 ดับ
}
```

และจากการทำงานโปรแกรมตัวอย่างนี้จะเห็นได้ว่าแม้จะสั่งให้ for วนทำงานซ้ำ 50 รอบ แต่ในการทำงานแต่ละรอบนั้นมีคำสั่ง if ไว้คอยตรวจสอบ ว่าถ้าหาก i มีค่าเท่ากับ 10 ให้กระโดดการทำงานไปที่ led2_ON ด้วยคำสั่ง if(i==10) goto led2_ON; ดังนั้นการทำงานที่เขียนสั่งให้ led1 ติดเป็นเวลา 5 วินาทีจะไม่เกิดขึ้นเลยถึงแม้จะเขียนคำสั่งเอาไว้ก็ตาม เพราะเมื่อทำงานถึงบรรทัดสุดท้ายโปรแกรมก็จะกลับไปเข้าทำงานในลูป for ใหม่และเริ่มนับใหม่จาก 0 อีกครั้ง และเมื่อนับไปถึง 10 ก็กระโดดข้ามไปทำงานที่ led2_ON อีก และวนทำงานอยู่อย่างนี้ตลอดไป และหากลองทำการส่งค่า i ไปแสดงที่ Serial monitor ก็จะเห็นได้ชัดว่าจะนับถึง 0 – 10 เท่านั้น ถึงแม้จะเขียนไว้ให้วนทำงานซ้ำถึง 50 รอบ แต่การทำงานจริงจะนับถึงแค่ 10 เท่านั้น

```
#define led1 8                          //ตั้งชื่อ led1 ขึ้นมาใช้แทน ขา 8
#define led2 9                          //ตั้งชื่อ led2 ขึ้นมาใช้แทน ขา 9
void setup() {
  pinMode(led1,OUTPUT);                 //ให้ขา 8 ทำงานเป็นเอาต์พุต
  pinMode(led2,OUTPUT);                 //ให้ขา 9 ทำงานเป็นเอาต์พุต
  Serial.begin(9600);                   //กำหนดความเร็วในการรับ-ส่งข้อมูล
}
void loop() {
   for(int i=0;i<50;i++){               //วนลูพนับเพิ่มค่า i จำนวน 50 รอบ
    Serial.println(i);                  //ส่งค่าข้อมูล i ออกทาง Serial Port
    if(i == 10) goto led2_ON;           //ถ้า i เท่ากับ 10 ให้กระโดดไปที่ led2_ON
       digitalWrite(led1,HIGH);         //ให้ led1 ติด
       delay(200);                      //หน่วงเวลา 200 มิลลิวินาที
       digitalWrite(led1,LOW);          //ให้ led1 ดับ
       delay(200);                      //หน่วงเวลา 200 มิลลิวินาที
   }
       digitalWrite(led1,HIGH);         //ให้ led1 ติด
       delay(5000);                     //หน่วงเวลา 5 วินาที
   led2_ON: digitalWrite(led2,HIGH);    //ให้ led2 ติด
            delay(500);                 //หน่วงเวลาครึ่งวินาที
            digitalWrite(led2,LOW);     //ให้ led2 ดับ
}
```



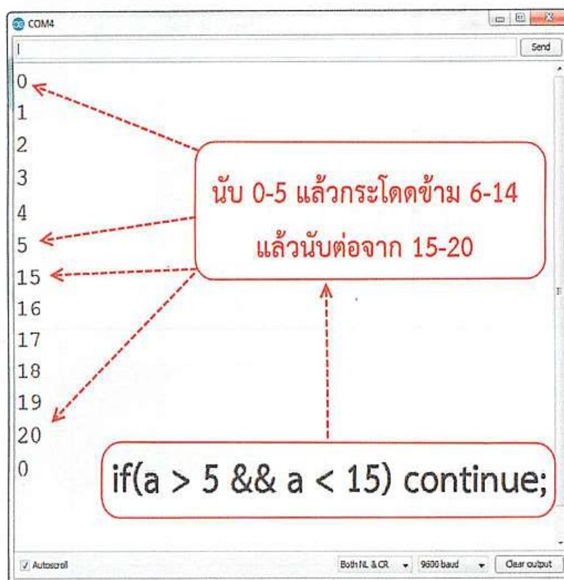


คำสั่ง continue

continue เป็นคำสั่งที่ใช้สำหรับการข้ามการทำงานบางช่วงของการวนลูป ใช้สำหรับลูป for ,while ,do...while แล้วแต่จะกำหนดว่าจะให้ข้ามการทำงานช่วงใด ให้ผู้อ่านทดลองเขียนโปรแกรมตามตัวอย่างเพื่อดูค่าข้อมูลที่ส่งไปแสดงที่ serial monitor เพื่อให้เข้าใจหลักการทำงานได้โดยง่าย

```
void setup() {
  Serial.begin(9600);                //กำหนดความเร็วในการรับ-ส่งข้อมูล

}
void loop() {
  for(int a = 0; a <= 20; a++){      //วนลูพให้เพิ่มค่า a ขึ้น 20 รอบ
    if(a > 5 && a < 15) continue;    //ถ้า a มากกว่า 5 และ a น้อยกว่า 15
                                     //ให้ใช้คำสั่ง continue
       delay(500);                   //หน่วงเวลาครึ่งวินาที
       Serial.println(a);            //ส่งค่า a ไปแสดงผลที่ Serial monitor
  }
}
```



จากโปรแกรมตัวอย่างนี้จะเห็นว่าถึงแม้ว่าในลูป for นั้นจะสั่งงานให้วนทำงานซ้ำ 20 รอบ และสั่งให้ส่งค่าข้อมูล a ออกไปแสดงผลทุกๆ ครึ่งวินาที แต่จากการทำงานของ if(a < 5 && a < 15) continue ดังนั้นเมื่อเราสั่งให้ส่งค่า a ไปแสดงผลที่ Serial monitor จะพบว่าค่าตัวเลข 6-14 นั้นหายไป ที่เป็นเช่นนี้ก็เพราะผลการทำงานของคำสั่ง continue นั่นเอง เมื่อ a มากกว่า 5 และ a น้อยกว่า 15 ให้ continue ดังนั้นค่าตัวเลขที่มากกว่า 5 และน้อยกว่า 15 ก็คือตัวเลข 6-14 จะไม่ถูกส่งไปแสดงผล แต่การวนลูป for นั้นจะยังคงทำงานตามปกติ นอกจากจะใช้คำสั่ง continue ในการข้ามช่วงค่าตัวเลขตรงกลางไปได้แล้วยังสามารถสั่งให้ข้ามช่วงแรก หรือ ช่วงสุดท้ายก็ได้

```
void setup() {
      Serial.begin(9600);
}

void loop() {
      for(int a=0;a<=20;a++){
       if(a > 10) continue;
       delay(500);
       Serial.println(a);
      }
}
//ตัวอย่างนี้โปรแกรมจะส่งค่าการนับ 0-10 เท่านั้น
```

↔

```
void setup() {
      Serial.begin(9600);
}

void loop() {
      for(int a=0;a<=20;a++){
       if(a < 10) continue;
       delay(500);
       Serial.println(a);
      }
}
//ตัวอย่างนี้โปรแกรมจะส่งค่าการนับ 10-20 เท่านั้น
```

Variables ค่าคงที่ และ ตัวแปร

ใน Arduino นั้น ค่าคงที่และตัวแปร และชนิดของข้อมูลต่างๆถูกจัดรวมไว้อยู่ในหมวดหมู่เดียวกัน

Constants ค่าคงที่

HIGH	เป็นค่าสูงสุดทางดิจิทัล มีค่าเป็น 1 หรือ มีไฟ
LOW	เป็นค่าต่ำสุดทางดิจิทัล มีค่าเป็น 0 หรือ ไม่มีไฟ
INPUT	เป็นค่าการทำงานของพอร์ต ที่มีการทำงานเป็นอินพุต ใช้รับสัญญาณเข้า
OUTPUT	เป็นค่าการทำงานของพอร์ต ที่มีการทำงานเป็นเอาต์พุต ใช้ส่งสัญญาณออก
INPUT_PULLUP	เป็นการสั่งเปิดใช้งาน R pull up ภายในตัวไอซี
LED_BUILTIN	เป็นพอร์ตที่ต่ออยู่กับหลอด LED ของบอร์ด Arduino ที่มีมาให้
true	เป็นค่าทางตรรกะ มีค่าเป็น จริง
false	เป็นค่าทางตรรกะ มีค่าเป็น เท็จ

HIGH และ LOW

HIGH และ LOW เป็นค่าคงที่ ที่มีค่าของตัวเองที่แน่นอนเปลี่ยนแปลงไม่ได้เป็นค่าที่ เมื่อประกาศค่าไว้อย่างไรก็จะเป็นค่านั้น HIGH หมายถึงสถานะลอจิก 1 หมายถึงมีไฟ และ LOW หมายถึงสถานะ ลอจิก 0 หมายถึงไม่มีไฟ ส่วนใหญ่จะใช้งานเพื่อส่งสัญญาณออกทางขาเอาต์พุตว่าจะให้ขานั้นมีสถานะ เป็น HIGH มีไฟหรือ เป็น LOW ไม่มีไฟ HIGH และ LOW นี้มักใช้กับฟังก์ชันที่ส่งค่าออกเป็นข้อมูลดิจิทัลคือฟังก์ชันที่ชื่อว่า digitalWrite() ใช้สำหรับสั่งงานออกเป็น OUTPUT เช่นสั่งให้ LED ติดหรือดับ หรือนำไปประยุกต์ใช้สั่งงานควบคุมรีเลย์ เพื่อสั่งเปิด-ปิดไฟ หรือการขับมอเตอร์ และอื่นๆอีกมากมาย

แต่ก่อนที่จะสั่งงานให้เป็น HIGH หรือเป็น LOW นั้น จะต้องใช้ฟังก์ชัน pinMode() เพื่อประกาศให้ขานั้นๆทำงานเป็นเอาต์พุตเสียก่อน จึงจะใช้ฟังก์ชัน digitalWrite() ได้

```
digitalWrite(8,HIGH);  //สั่งให้ขา 8 มีไฟ
digitalWrite(8,LOW);   //สั่งให้ขา 8 ไม่มีไฟ
```

ตัวอย่างโปรแกรมที่ใช้งานจริง

```
void setup() {              //เริ่มฟังชั่น setup
 pinMode(8,OUTPUT);         //กำหนดให้ขา 8 เป็นเอาต์พุต
 digitalWrite(8,HIGH);      //สั่งให้ขา 8 มีสถานะเป็น HIGH
}                           //จบฟังชั่น setup
void loop() {

}
```

โปรแกรมตัวอย่างนี้จะทำให้หลอด LED ที่ต่ออยู่กับขา 8 ติดสว่างอยู่อย่างนั้น

INPUT และ OUTPUT

INPUT และ OUTPUT เป็นค่าคงที่ ที่ใช้สำหรับกำหนดสถานะ โหมดการทำงานของ ขาต่างๆของ Arduino ว่าจะให้ขาใดทำงานเป็นอินพุต หรือจะใช้ขาใดทำงานเป็นเอาต์พุตก็ได้ การสั่งงานนั้นจะใช้กับฟังก์ชัน pinMode() โดยกำหนดขาและค่าคงที่ INPUT หรือ OUTPUT ลงในวงเล็บ

```
pinMode(7,INPUT);     //สั่งให้ขา 7 ทำงานในโหมดอินพุต
pinMode(8,OUTPUT);    //สั่งให้ขา 8 ทำงานในโหมดเอาต์พุต
```

ตัวอย่างโปรแกรมใช้งานจริง

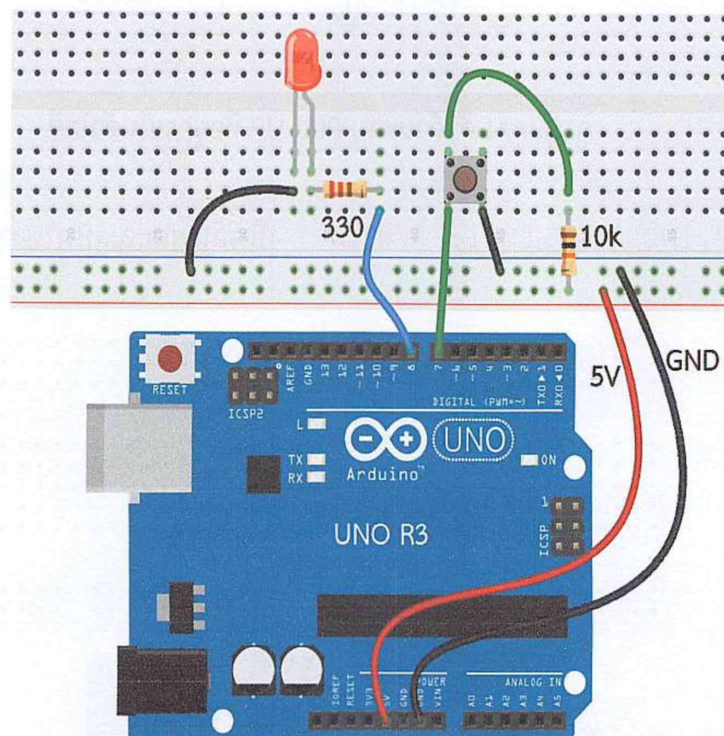
```
void setup() {              //เริ่มฟังชั่น setup
 pinMode(7,INPUT);          //กำหนดให้ขา 7 เป็นอินพุต
 pinMode(8,OUTPUT);         //กำหนดให้ขา 8 เป็นเอาต์พุต
}                           //จบฟังชั่น setup
void loop() {               //เริ่มฟังชั่น loop

   if(digitalRead(7)==0)         digitalWrite(8,HIGH);
//ถ้าหากอ่านค่าจากขา 7 ได้เท่ากับ 0 //สั่งให้ขา 8 มีสถานะเป็น HIGH

   else     digitalWrite(8,LOW);
//นอกจากนั้น //สั่งให้ขา 8 มีสถานะเป็น LOW
}                           //จบฟังชั่น loop
```

การทำงานของโปรแกรมนี้จะใช้งานขา 7 เป็นอินพุต รับค่าจากการกดสวิตช์ลงกราวด์ โดยการใช้ R ค่า 10k ต่อรับไฟ 5V เข้ามาที่ขา7 การทำเช่นนี้จะทำให้ขา 7 มีสถานะเป็น HIGH เรียกว่าอินพุตแบบพูลอัพ

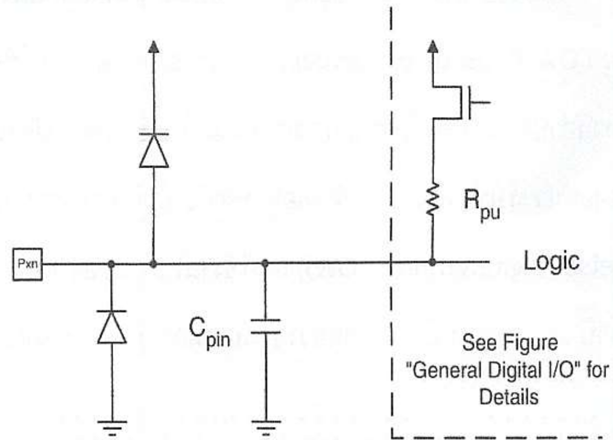
(PULL UP) และขาข้างหนึ่งของสวิตช์นั้นต่ออยู่กับขา 7 และอีกขาต่ออยู่กับกราวด์ เมื่อมีการกดสวิตช์จะทำให้ขา 7 มีสถานะเป็น LOW คือไม่มีไฟ โปรแกรมจะตรวจสอบสถานะด้วยการใช้คำสั่ง **if(digitalRead(7)==0)** หมายความว่า ถ้าหากอ่านค่าจากขา7 ได้เป็น 0 หมายถึงไม่มีไฟ ก็จะสั่งให้ **digitalWrite(8,HIGH);** ก็คือการสั่งให้ขา 8 นั้นจ่ายไฟออกมา ทำให้หลอด LED ที่ต่ออยู่กับขา 8 นั้นติดสว่างขึ้น และเมื่อปล่อยสวิตช์หลอดไฟก็จะดับลง ด้วยการใช้ **else digitalWrite(8,LOW);** การใช้งานในส่วนของ INPUT และ OUTPUT ก็มีหลักการใช้งานที่ง่ายๆเพียงเท่านี้ และรูปต่อไปนี้เป็นการต่อวงจรเพื่อทดลองการทำงานของโปรแกรมนี้



****

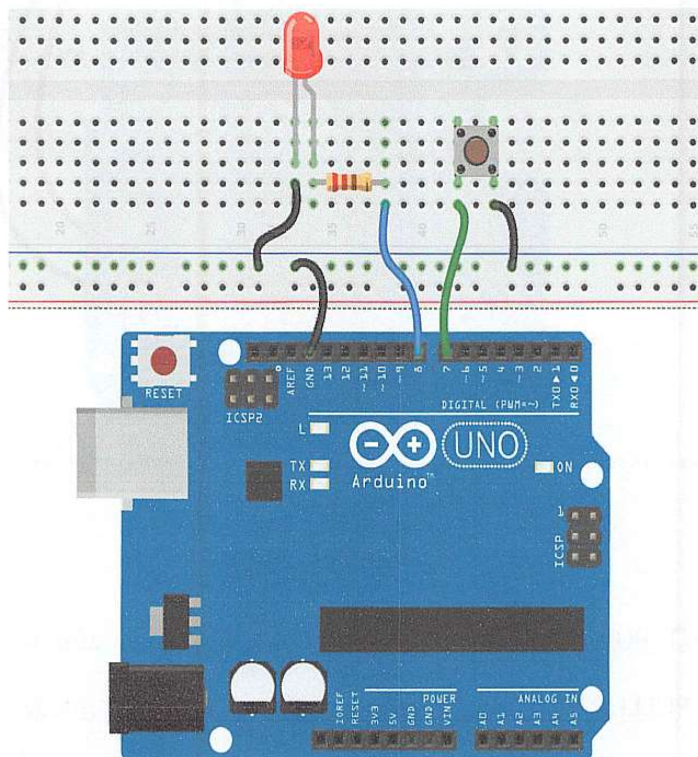
INPUT_PULLUP

INPUT_PULLUP เป็นการตั้งค่าเปิดการใช้งานตัวต้านทานพูลอัพ ที่อยู่ภายในตัวไอซี โดยเมื่อเรียกใช้ INPUT_PULLUP แล้วจะทำให้ขานั้นมีสถานะเป็น HIGH ใช้สำหรับการสั่งทำงานในโหมดอินพุต แบบพูลอัพ เพื่อรับค่าจากการกดสวิตช์ต่อลงกราวด์ หรือการกระทำใดใดที่จะทำให้ขานั้นถูกต่อลงกราวด์ จะทำให้ไมโครคอนโทรลเลอร์รู้ว่ามีการเปลี่ยนแปลงเกิดขึ้น เราจึงสามารถนำค่าการเปลี่ยนแปลงนี้ไปเป็นเงื่อนไขในการเขียนสั่งงานต่างๆ เช่น หากสถานะเกิดเป็น LOW ก็สั่งให้เปิดไฟ เป็นต้น



ภาพแสดง ตัวต้านทาน Pull UP ที่อยู่ภายในตัวไอซี

```
pinMode(7,INPUT_PULLUP); //สั่งให้ 7 เป็นอินพุตและเปิดใช้งาน R PULLUP
```



ข้อดีของ INPUT_PULLUP ก็คือ ในการใช้งานเป็นอินพุตแบบพูลอัพนั้น เราไม่ต้องต่อ R PULLUP ไว้ข้างนอก เราสามารถต่อขานั้นเข้าสวิตซ์เพื่อกดลงกราวด์ได้ทันดี เพราะเมื่อสั่ง INPUT_PULLUP แล้วจะเป็นการสั่งให้ต่อใช้งาน R ที่ทำหน้าที่ PULLUP ที่อยู่ภายในตัวไอซี เราจึงไม่ต้องต่อ R เพิ่มเติมแต่อย่างใด

```
ตัวอย่างโปรแกรมใช้งานจริง

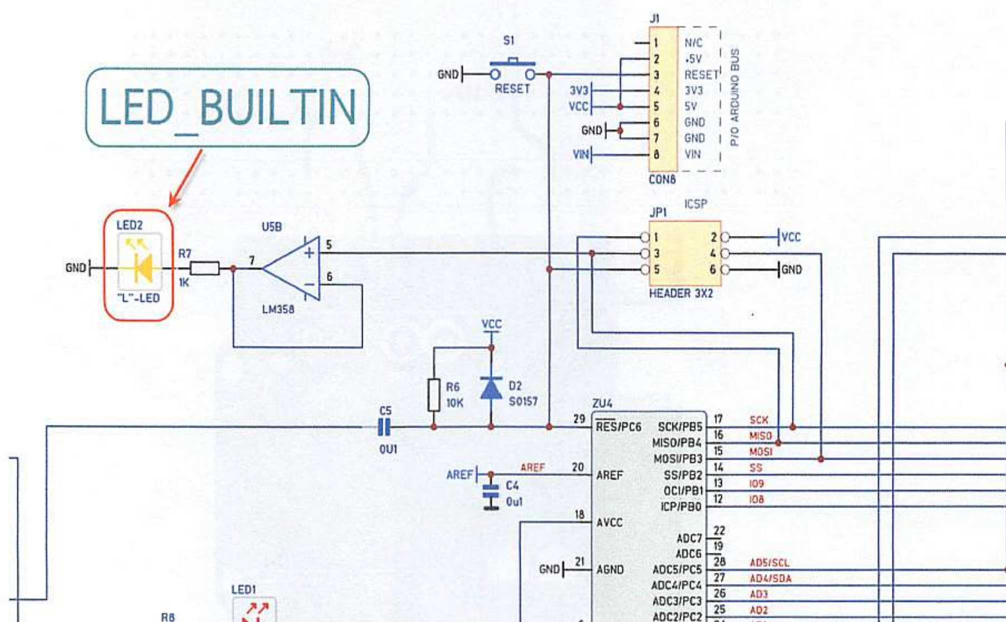
void setup() {                //เริ่มฟังชั่น setup
 pinMode(7,INPUT_PULLUP);     //กำหนดให้ขา 7 เป็นอินพุต
 pinMode(8,OUTPUT);           //กำหนดให้ขา 8 เป็นเอาต์พุต
}                             //จบฟังชั่น setup
void loop() {                 //เริ่มฟังชั่น loop

   if(digitalRead(7)==0)        digitalWrite(8,HIGH);
//ถ้าหากอ่านค่าจากขา 7 ได้เท่ากับ 0 //สั่งให้ขา 8 มีสถานะเป็น HIGH

   else    digitalWrite(8,LOW);
//นอกจากนั้น //สั่งให้ขา 8 มีสถานะเป็น LOW
}                             //จบฟังชั่น loop
```

LED_BUILTIN

LED_BUILTIN เป็นค่าคงที่ ที่หมายถึงขาที่ต่ออยู่กับหลอด LED ที่อยู่ในบอร์ด Arduino ซึ่งในบอร์ด UNO R3 นั้นจะอยู่ที่ขา 13 ดังนั้น LED_BUILTIN จึงหมายถึงหลอด LED ที่บอร์ด Arduino ใส่มาให้แล้วสามารถเรียกสั่งงาน ให้หลอดติดหรือดับ ได้ทันทีโดยไม่ต้องไปหาซื้อหลอด LED มาต่อเพิ่ม



ตัวอย่างการเขียนสั่งงาน LED_BUILTIN

```
void setup() {                          //เริ่มฟังชั่น setup

  pinMode(LED_BUILTIN,OUTPUT);          //ให้ LED_BUILTIN เป็นเอาต์พุต

}                                       //จบฟังชั่น setup

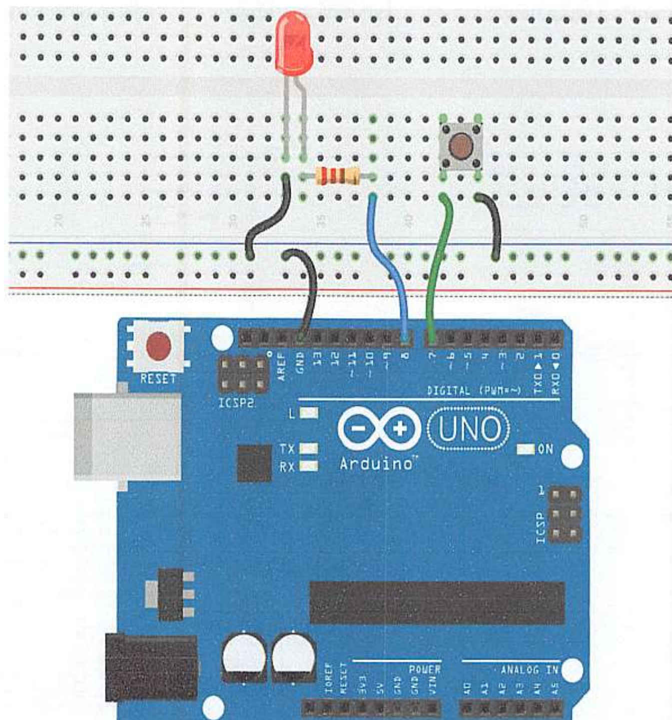
void loop() {                           //เริ่มฟังชั่น loop

  digitalWrite(LED_BUILTIN,HIGH);       //สั่งให้หลอด LED_BUILTIN ติด
  delay(500);                           //หน่วงเวลา ครึ่งวินาที
  digitalWrite(LED_BUILTIN,LOW);        //สั่งให้หลอด LED_BUILTIN ดับ
  delay(500);                           //หน่วงเวลา ครึ่งวินาที

}                                       //จบฟังชั่น loop
```

true และ false

true และ false เป็นค่าคงที่ทางตรรกะที่มีค่าเป็นไปได้เพียงสองค่าเท่านั้น คือ true หมายถึง จริงและ false หมายถึง เท็จ ส่วนใหญ่มักใช้กับการเปรียบเทียบค่าตัวแปรชนิด **boolean**

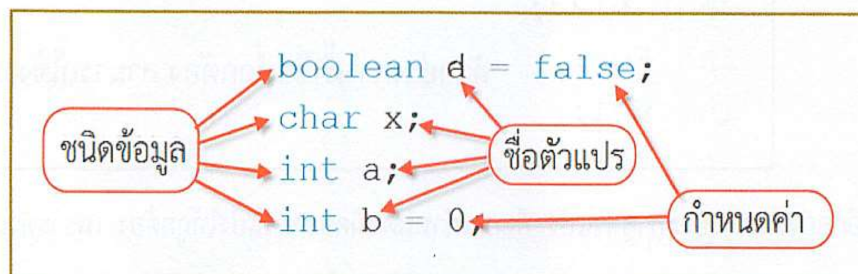


สำหรับในการทดลองนี้เราจะทดสอบการใช้งาน true และ false ด้วยการใช้สวิตซ์ BUTTON กดสั่งให้หลอด LED ติด และใช้สวิตซ์ตัวเดียวกันนี้ กดสั่งให้ดับ เป็นการสั่งงาน เปิด-ปิด ไฟด้วยสวิตซ์เพียงตัวเดียว ด้วยการตรวจสอบค่าทางตรรกะ true และ false

```
boolean a = false;              //สร้างตัวแปร boolean
void setup() {                  //เริ่มฟังชั่น setup
  pinMode(7,INPUT_PULLUP);      //ให้ขา 7 ทำงานเป็นอินพุต
  pinMode(8,OUTPUT);            //ให้ขา 8 ทำงานเป็นเอาต์พุต
}                               //จบฟังชั่น setup
void loop() {                   //เริ่มฟังชั่น loop
  if(digitalRead(7)==0){        //ถ้าหากอ่านค่าขา 7 ได้มีค่าเป็น 0
    delay(300);                 //หน่วงเวลาตรวจสอบว่ากดสวิตซ์จริง
    a = !a;                     //ให้กลับสถานะค่าลอจิกของ a
    if(a == true) digitalWrite(8,HIGH); //ถ้า a มีค่าเป็น true ให้หลอดติด
    else  digitalWrite(8,LOW);          //นอกเหนือจากนั้นให้หลอดดับ
  }
}                                       //จบฟังชั่น loop
```

ตัวแปร (Variables)

ตัวแปรเป็นสิ่งที่ใช้ในการเก็บข้อมูล เพื่อใช้งานในการอ้างอิง หรือการคำนวณต่าง ๆ ในโปรแกรม สิ่งที่เรียกว่าตัวแปรนั้น คือคำหรือชื่อเรียกต่าง ๆ ที่เราต้องสร้างขึ้นมา และตั้งชื่อขึ้นมาเองเพื่อใช้ในการเก็บข้อมูลต่าง แล้วแต่การใช้งาน แต่การตั้งชื่อตัวแปรนั้นมีกฎในการตั้งชื่อ ต้องตั้งให้ถูกต้องตามกฎ ในการประกาศตัวแปรในภาษา C นั้นมีหลักการ คือ ต้องกำหนดชนิดของข้อมูลก่อน แล้วตามด้วยชื่อที่เราตั้งขึ้นมาเอง แล้วปิดท้ายด้วยเครื่องหมายเซมิโคลอน ;



นอกจากจะกำหนดชนิดและตั้งชื่อขึ้นมาแล้ว เรายังสามารถกำหนดค่าเริ่มต้นขึ้นมาได้อีกด้วยว่าจะให้มีค่าเท่าใด หากไม่มีการกำหนดค่าเริ่มต้นให้กับตัวแปร ภาษา C จะกำหนดค่าเริ่มต้นให้เป็น 0 เองโดยอัตโนมัติ เพียงแต่การตั้งชื่อนั้น ต้องตั้งชื่อให้ถูกต้องตามกฎอย่างเคร่งครัด กฎในการตั้งชื่อมีดังนี้

1. การตั้งชื่อให้ใช้ตัวอักษร A ถึง Z รวมไปถึงตัวเลข 0 ถึง 9 และเครื่องหมายขีดล่าง _ Underscore ได้ แต่ชื่อนั้นต้องขึ้นต้นด้วยตัวอักษร เท่านั้นจะขึ้นต้นด้วยตัวเลขไม่ได้ ยกตัวอย่างเช่น **1vr** แบบนี้ถือว่าผิดกฎการตั้งชื่อ เมื่อทำการคอมไพล์โปรแกรมจะเออเร่อทันที ต้องแก้ไขให้เป็น **vr1** แบบนี้ถือว่าถูกกฎระเบียบจึงจะสามารถใช้งานได้

2. สามารถตั้งชื่อตัวแปรได้ยาวไม่เกิน 31 ตัวอักษรเท่านั้นซึ่งในความเป็นจริงคงไม่มีโปรแกรมเมอร์คนใดที่จะตั้งชื่อตัวแปรได้ยาวขนาดนั้นอยู่แล้ว

3. การตั้งชื่อตัวแปรนั้นจะต้องไม่ซ้ำกับชื่อที่เป็นคำสงวนของโปรแกรม คำสงวนหมายความว่าเป็นคำที่เป็นคำสั่งในการสั่งงานของภาษา C เช่น for , if , while คำเหล่านี้เป็นชุดคำสั่งในภาษา C จึงไม่สามารถนำมาตั้งเป็นชื่อตัวแปรได้ แต่ภาษา C นั้นเป็น Case Sensitive หมายความว่าตัวอักษรพิมพ์ใหญ่และตัวอักษรพิมพ์เล็กนั้น ถือว่าเป็นคนละตัวกัน ดังนั้นหากเราตั้งชื่อตัวแปรว่า For จึงไม่ซ้ำกับคำสั่ง for แต่อย่างใด เพราะภาษา C ถือว่าเป็นคนละตัวกัน

```
char 123b;
int for;
int 1v;
```

การตั้งชื่อที่ไม่ถูกต้อง ไม่สามารถใช้งานได้

ตัวอย่างการตั้งชื่อตัวแปรที่ไม่ถูกต้อง

```
char b123;
int For;
int v_1;
```

ตัวอย่างการตั้งชื่อที่ถูกต้อง สามารถใช้งานได้

สิ่งสำคัญอีกอย่างในการประกาศตัวแปร คือการกำหนดชนิดของตัวแปรให้ถูกต้อง เพราะตัวแปรแต่ละชนิดนั้นจะมีขอบเขต ในการเก็บข้อมูลที่ไม่เท่ากัน ควรเลือกใช้ให้ถูกต้องเช่นหากต้องการเก็บข้อมูลชนิดตัวเลขที่มีจุดทศนิยมต้องใช้ตัวแปรชนิดที่เก็บเลขทศนิยมได้เท่านั้นคือ float หรือ double หากประกาศใช้ตัวแปรเป็น int

แต่สั่งให้เก็บข้อมูลตัวเลขที่มีจุดทศนิยม หากนำค่าข้อมูลไปประมวลผลจะได้ค่าที่ผิดเพี้ยนไป การใช้ตัวแปรผิดประเภท จะทำให้โปรแกรมที่เขียนขึ้นมานั้นประมวลผลข้อมูลผิดพลาดได้

```
float a = 5.5;
float b = 5.2;
int v;
      v = a + b;
```

v จะมีค่าเป็น 10 ไม่ใช่ 10.7
เพราะว่า int ไม่สามารถเก็บตัวเลขที่มีจุดทศนิยมได้

หรือ การประกาศตัวแปรที่เก็บข้อมูลได้จำนวนมากเช่นชนิด long เพื่อนำมาเก็บค่าตัวเลข แค่ 0-100 เท่านั้น ก็จะทำให้สูญเสียพื้นที่ในหน่วยความจำโดยไปไม่จำเป็น เพราะในการประกาศตัวแปรนั้นจะเป็นการจองพื้นที่ในหน่วยความจำเพื่อเก็บข้อมูลนั้น เปรียบได้กับการเหมารถสิบล้อ เพียงเพื่อที่จะไปซื้อกับข้าวที่ตลาด ซึ่งจริง ๆ แล้วปั่นจักรยานไปก็ได้

เพื่อความเข้าใจที่ชัดเจนให้ผู้อ่านทดลองเขียนโปรแกรมตามตัวอย่างและกดดูผลลัพธ์ได้ที่ Serial Monitor

Serial | Arduino 1.8.5

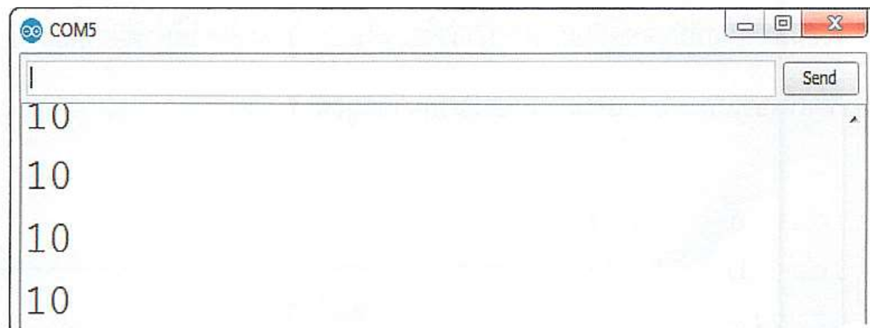
File Edit Sketch Tools Help

Serial §

```
float a = 5.5;          //สร้างตัวแปร a ชนิดเลขทศนิยมให้มีค่า 5.5
float b = 5.2;          //สร้างตัวแปร b ชนิดเลขทศนิยมให้มีค่า 5.2
int v;                  //สร้างตัวแปร v ชนิดเลขจำนวนเต็ม
void setup() {
 Serial.begin(9600);
  v = a + b;            //นำค่าผลบวกของ a และ b มาเก็บใน v
}

void loop() {
 Serial.println(v);     //แสดงค่าผลบวกออกทางมอนิเตอร์
 delay(1000);
}
```

Done uploading.



จากโปรแกรมตัวอย่างนี้จะเห็นว่าตัวแปร a และ b นั้นเป็นตัวแปรชนิดเลขทศนิยม float และเมื่อนำค่าของ a และ b มาบวกกัน ก็หมายถึง 5.5+5.2 ซึ่งผลลัพธ์จริง ๆ นั้นคือ 10.7 แต่ในโปรแกรมนั้นใช้ตัวแปร v ซึ่งเป็นตัวแปรชนิดชนิดเลขจำนวนเต็ม int มาเก็บผลลัพธ์ ซึ่งตัวแปรชนิด int นี้ไม่สามารถเก็บตัวเลขที่ไม่ใช่เลขจำนวนเต็มได้ มันจะเก็บได้เฉพาะตัวเลขจำนวนเต็ม เท่านั้น ดังนั้นเมื่อเราสั่งแสดงค่าออกมาที่ Serial monitor จะเห็นว่าค่าของตัวแปร v ที่เก็บได้คือ 10 เท่านั้น ไม่ใช่ 10.7 นี่เป็นตัวอย่างของการใช้งานตัวแปรผิดประเภท จะเห็นได้ชัดว่าผลลัพธ์ที่ได้นั้นผิดเพี้ยนไปจากความเป็นจริง ดังนั้นจึงต้องจำเอาไว้ให้ดีว่าการประกาศตัวแปรนั้น ต้องใช้ให้ถูกต้องและสอดคล้องกับชนิดข้อมูลที่ต้องการเก็บ ดังนั้นจากโปรแกรมตัวอย่างนี้ หากต้องการให้แสดงผลลัพธ์ที่แท้จริงและถูกต้องจะต้องประกาศตัวแปร v ให้เป็นชนิดที่เก็บเลขทศยมได้เช่นกัน float v; และเพื่อความเข้าใจที่ชัดเจนให้ผู้อ่านทดลองเปลี่ยนชนิดตัวแปรที่เก็บค่าผลลัพธ์ให้เป็น float v; แล้วอัพโหลดโปรแกรมดูผลลัพธ์ที่เกิดขึ้นจะช่วยทำให้เข้าใจได้ดีขึ้น

ตัวแปรชนิดอาร์เรย์ Array

ตัวแปรชนิดอาร์เรย์ Array เป็นการเก็บข้อมูลโดยใช้ตัวแปรเพียงตัวเดียว เก็บข้อมูลหลายๆค่าโดยการแบ่งพื้นที่แยกเก็บข้อมูลแต่ละค่าไว้ คล้ายๆกล่องเก็บของที่มีช่องเก็บของหลายๆช่อง โดยข้อมูลที่เก็บนั้นจะต้องเป็นข้อมูลชนิดเดียวกันเท่านั้น

ยกตัวอย่างเช่นหากเรามีข้อมูลตัวเลขอยู่ 9 ค่า ในการตั้งชื่อประกาศตัวแปรนั้นคงไม่ใช่ปัญหาเท่าใดนัก แต่จะต้องตั้งชื่อตัวแปรขึ้นมาถึง 9 ชื่อเพราะชื่อตัวแปรนั้นจะซ้ำกันไม่ได้

```
int v1 = 10;
int v2 = 20;
int v3 = 30;
int v4 = 40;
int v5 = 50;
int v6 = 60;
int v7 = 70;
int v8 = 80;
int v9 = 90;
```

แต่ถ้าหากมีข้อมูลเป็น 100 ค่า การที่จะมานั่งตั้งชื่อตัวแปรเป็น 100 ชื่อคงไม่ใช่เรื่องง่าย ตัวแปรแบบอาร์เรย์จะทำให้เรื่องนี้ง่ายขึ้น เพราะเป็นการประกาศตัวแปรโดยใช้ชื่อตัวแปรเพียงชื่อเดียวแต่สามารถเก็บข้อมูลได้หลายค่า โดยค่าของข้อมูลแต่ละค่านั้นจะถูกแยกเก็บในแต่ละตำแหน่งเรียงกันไป การประกาศตัวแปรชนิดอาร์เรย์จะใช้เครื่องหมาย [ ] เป็นตัวบอกว่านี่คือตัวแปรแบบอาร์เรย์

```
int v[]={10,20,30,40,50,60,70,80,90};
```

ตัวอย่างแสดงให้เห็นว่าตัวแปรชื่อ v เพียงตัวเดียว นั้นเก็บค่าตัวเลขไว้ทั้งหมด 9 ค่า

ค่าข้อมูลจะเขียนไว้ในวงเล็บปีกกา และข้อมูลแต่ละค่าจะต้องมีเครื่องหมาย , กั้นไว้เป็นตัวบ่งบอกว่าค่าของข้อมูลแต่ละค่าในวงเล็บปีกกานั้นอยู่คนละตำแหน่งกัน โดยตำแหน่งแรกที่อยู่ทางซ้ายมือสุดคือตำแหน่งเริ่มต้นที่ 0 การเรียกใช้ข้อมูลนั้นจะมีตำแหน่งเป็นตัวชี้ ว่าต้องการใช้ข้อมูลตำแหน่งใด

ตำแหน่งข้อมูล
0 1 2 3 4 5 6 7 8

```
int a;
int v[]={10,20,30,40,50,60,70,80,90};
    a = v[4];
```

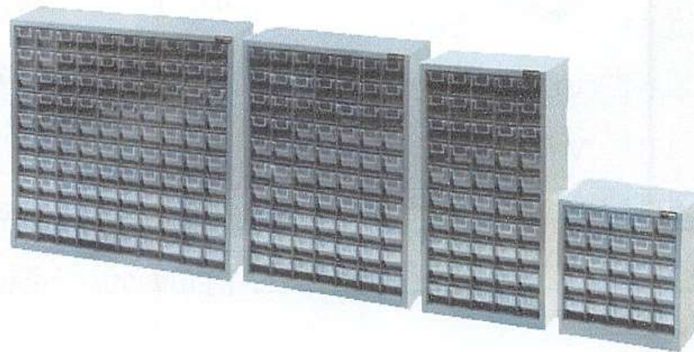
ตัวอย่างนี้ a จึงมีค่าข้อมูลในตำแหน่งที่ 4 คือ 50

ข้อมูล →	10	20	30	40	50	60	70	80
ตำแหน่ง →	0	1	2	3	4	5	6	7

ภาพตัวอย่างการเก็บข้อมูลชนิดอาร์เรย์

นอกจากนี้ยังมีตัวแปรอาร์เรย์แบบ 2 มิติ และ 3 มิติ แต่ในเบื้องต้นนั้นเพียงแค่นี้ก็เพียงพอต่อการเริ่มต้นเขียนโปรแกรมกันแล้ว แล้วเราค่อยศึกษาเพิ่มเติมกันต่อไป

หากจะเปรียบเทียบเพื่อความเข้าใจที่ง่ายขึ้นก็คล้ายๆกับตู้เก็บของที่มีช่องเก็บได้หลายช่องนั่นเอง



ให้ทดลองเขียนโปรแกรมตัวอย่างต่อไปนี้ เพื่อทดสอบดูผลการทำงานให้เข้าใจได้ดียิ่งขึ้น โดยโปรแกรมต่อไปนี้จะสร้างตัวแปรชนิดอาร์เรย์ชื่อว่า data ให้มีข้อมูลตัวเลขทั้งหมด 10 ข้อมูลคือ 5,10,15,20,25,30,35,40,45,50 และใช้ลูป for วนนับเพิ่มค่าตัวแปร i จำนวน 10 รอบ และสั่งให้ส่งข้อมูลนั้นไปแสดงที่ Serial monitor โดยอ้างอิงจากตำแหน่งของข้อมูล ตามการนับของตัวแปร i ในการนับและการส่งข้อมูลในแต่ละครั้งจะส่งผลให้ led กระพริบด้วย 1 ครั้ง โดยให้ต่อหลอด LED ไว้ที่ขา 8 และให้สังเกตผลลัพธ์ที่ส่งไปแสดงที่จอ Serial monitor ที่เครื่องคอมพิวเตอร์

//ตัวอย่างโปรแกรมการใช้งานตัวแปรชนิดอาร์เรย์

```
#define led 8
int data[]={5,10,15,20,25,30,35,40,45,50}; //สร้างตัวแปรอาร์เรย์
void setup() {
  pinMode(led,OUTPUT);          //ให้ขา 8 ทำงานเป็นเอาต์พุต
  Serial.begin(9600);           //กำหนดความเร็วในการรับ-ส่งข้อมูล 9600 บิตต่อวินาที
}
void loop() {
  for(int i=0;i<10;i++){        //วนนับเพิ่มค่าตัวแปร i จำนวน 10 รอบ
    Serial.println(data[i]);    //ส่งข้อมูลตัวแปรอาร์เรย์จากตำแหน่งการนับ i
    digitalWrite(led,HIGH);     //ให้ LED ติด แสดงผลการส่งข้อมูล
    delay(500);                 //หน่วงเวลาครึ่งวินาที (500ms)
    digitalWrite(led,LOW);      //ให้ LED ดับ
    delay(1500);                //หน่วงเวลา 1.5 วินาที (1500ms)
  }
}
```

```
int data[]={5,10,15,20,25,30,35,40,45,50};
```

data[0] หมายถึงข้อมูลในตัวแปร data ตำแหน่ง 0 คือเลข 5

data[1] หมายถึงข้อมูลในตัวแปร data ตำแหน่ง 1 คือเลข 10

data[2] หมายถึงข้อมูลในตัวแปร data ตำแหน่ง 2 คือเลข 15

data[3] หมายถึงข้อมูลในตัวแปร data ตำแหน่ง 3 คือเลข 20

data[4] หมายถึงข้อมูลในตัวแปร data ตำแหน่ง 4 คือเลข 25

data[5] หมายถึงข้อมูลในตัวแปร data ตำแหน่ง 5 คือเลข 30

data[6] หมายถึงข้อมูลในตัวแปร data ตำแหน่ง 6 คือเลข 35

data[7] หมายถึงข้อมูลในตัวแปร data ตำแหน่ง 7 คือเลข 40

data[8] หมายถึงข้อมูลในตัวแปร data ตำแหน่ง 8 คือเลข 45

data[9] หมายถึงข้อมูลในตัวแปร data ตำแหน่ง 9 คือเลข 50

Data Type ชนิดของข้อมูล

ชนิด		ใช้พื้นที่
void	ใช้ในการประกาศฟังก์ชัน เป็นการบอกว่าจะไม่ส่งข้อมูลใดออกไปจากฟังก์ชัน	
boolean	เป็นข้อมูลทางตรรกะ มีเพียงสองค่า คือ จริง true หรือ เท็จ false	1 bit
char	เป็นข้อมูลชนิด อักขระ ค่าของข้อมูล -127 ถึง 128	1 byte
unsigned char	ชนิดเดียวกับ char แต่จะไม่นับค่าลบ จะเก็บเพียงค่าบวก 0-255	1 byte
byte	ใช้เก็บข้อมูลที่ไม่มีค่าลบ เก็บได้ 8 บิต เก็บค่าได้ 0-255	1 byte
int	ใช้เก็บข้อมูลเลขจำนวนเต็ม -32768 ถึง 32,767	2 byte
unsigned int	ใช้เก็บข้อมูลเลขจำนวนเต็ม แต่ไม่นับค่าลบ เก็บได้ 0 - 65,535	2 byte
word	ในบอร์ด UNO จะเก็บค่าตัวเลขไม่นับค่าลบ แบบ 16 บิต	2 byte
long	เก็บตัวเลขแบบยาว มีค่าตั้งแต่ –2,1448,483,648 ถึง 2,147,483,647	4 byte
unsigned long	เก็บตัวเลขแบบยาวไม่นับค่าลบ มีค่าตั้งแต่ 0 – 4,294,967,295	4 byte
short	ในArduino short คือการเก็บข้อมูลชนิดสั้น 16 บิต –32768 ถึง 32767	2 byte
float	ใช้สำหรับเก็บข้อมูลตัวเลขที่มีจุดทศยม	4 byte
double	เหมือนกับ float แต่เก็บได้เยอะกว่า	8 byte
string (char array)	ใช้ในการเก็บข้อความและอักขระ	
String (object)	เป็น object	
array	เป็นการเก็บข้อมูลชนิดอาร์เรย์	

ชนิดของตัวแปรในภาษาซี

ชนิดของตัวแปร	ขนาด (bits)	ขอบเขต	ข้อมูลที่เก็บ
char	8	-128 ถึง 127	ข้อมูลชนิดอักขระ ใช้เนื้อที่ 1 byte
unsigned char	8	0 ถึง 255	ข้อมูลชนิดอักขระ ไม่คิดเครื่องหมาย
int	16	-32,768 ถึง 32,767	ข้อมูลชนิดจำนวนเต็ม ใช้เนื้อที่ 2 byte
unsigned int	16	0 ถึง 65,535	ข้อมูลชนิดจำนวนเต็ม ไม่คิดเครื่องหมาย
short	8	-128 ถึง 127	ข้อมูลชนิดจำนวนเต็มแบบสั้น ใช้เนื้อที่ 1 byte
unsigned short	8	0 ถึง 255	ข้อมูลชนิดจำนวนเต็มแบบสั้น ไม่คิดเครื่องหมาย
long	32	-2,147,483,648 ถึง 2,147,483,649	ข้อมูลชนิดจำนวนเต็มแบบยาว ใช้เนื้อที่ 4 byte
unsigned long	32	0 ถึง 4,294,967,296	ข้อมูลชนิดจำนวนเต็มแบบยาว ไม่คิดเครื่องหมาย
float	32	3.4*10e(-38) ถึง 3.4*10e(38)	ข้อมูลชนิดเลขทศนิยม ใช้เนื้อที่ 4 byte
double	64	3.4*10e(-308) ถึง 3.4*10e(308)	ข้อมูลชนิดเลขทศนิยม ใช้เนื้อที่ 8 byte
long double	128	3.4*10e(-4032) ถึง 1.1*10e(4032)	ข้อมูลชนิดเลขทศนิยม ใช้เนื้อที่ 16 byte

ฟังก์ชันสำเร็จรูป

Functions ฟังก์ชันสำเร็จรูป ที่ Arduino สร้างมาให้ใช้งานได้อย่างง่ายดาย

เพื่อให้การเขียนคำสั่งนั้นดูใกล้เคียงกับภาษาของมนุษย์ ทำให้ง่ายต่อการเข้าใจ

ฟังก์ชันในหมวดหมู่ของ Digital I/O เป็นฟังก์ชันที่เกี่ยวกับการสั่งงานแบบดิจิทัลมีอยู่ด้วยกัน 3 ฟังก์ชัน

ฟังก์ชัน pinMode

pinMode();

pinMode() ฟังก์ชันพินโหมด เป็นฟังก์ชันสำเร็จรูปที่ใช้การตั้งค่าโหมดการทำงานของขาต่างๆ ว่าจะให้ทำงานเป็นอินพุต เพื่อใช้รับสัญญาณเข้ามา หรือ จะใช้งานเป็นเอาต์พุต เพื่อส่งสัญญาณออกไปควบคุมอุปกรณ์ต่างๆฟังนั้นนี้มีการทำงานเพียงแค่สองโหมดเท่านั้น คือ INPUT และ OUTPUT วิธีการเขียนสั่งงานนั้นให้ใส่ ชื่อตำแหน่งขา และ สถานะ ที่ต้องการ ลงในวงเล็บ โดยคั่นด้วยเครื่องหมายจุลภาค

(ในหมวดของ INPUT นั้น อาจมีการใช้งาน INPUT_PULLUP)

การเขียนนั้นให้เขียนด้วยตัวอักษรพิมพ์เล็กยกเว้นตัว M ให้เขียนด้วยตัวอักษรพิมพ์ใหญ่ ต้องเขียนให้ถูกด้วย และ สถานะ INPUT หรือ OUTPUT นั้นต้องเขียนด้วยตัวอักษรพิมพ์ใหญ่ทั้งหมด ต้องจำให้ดี

และตามหลักการในเบื้องต้น จะต้องเขียนฟังก์ชัน pinMode() นี้เอาไว้ในฟังก์ชัน setup(); จึงจะใช้งานได้ แต่ในการใช้งานระดับสูงสำหรับผู้ที่มีความชำนาญในการเขียนโปรแกรม อาจจะใช้เทคนิคอื่นๆ และไม่จำเป็นต้องประกาศไว้ในฟังก์ชัน setup ก็ได้

```
pinMode(ชื่อขา,สถานะ);
```

```
pinMode(8,OUTPUT);
```

ตัวอย่างการสั่งงานให้ขา 8 ทำงานเป็นเอาต์พุต

```
pinMode(A0,INPUT);
```

ตัวอย่างการสั่งงานให้ขา A0 ทำงานเป็นอินพุต อาจจะนำไปใช้เพื่อรับค่าจากเซนเซอร์ต่างๆ

changnop | Arduino 1.8.4

File Edit Sketch Tools Help

changnop §

```
void setup() {

  pinMode(8,OUTPUT);
  pinMode(A0,INPUT);
}

void loop() {

}
```

Done Saving.

ภาพตัวอย่างการใช้ฟังก์ชัน pinMode() กำหนดขา A0 เป็นอินพุต และขา 8 เป็นเอาต์พุต

ฟังก์ชัน digitalWrite()

```
digitalWrite();
```

ฟังก์ชัน digitalWrite(); เป็นฟังก์ชันสำเร็จรูปที่ใช้ส่งข้อมูลดิจิทัลออกทางที่กำหนดโดยข้อมูลที่เป็นดิจิทัลนั้นเรารู้กันอยู่แล้วว่ามีอยู่เพียงสองค่าเท่านั้น คือ 0 และ 1 โดย Arduino นั้นได้กำหนดชื่อค่าคงที่ขึ้นมา แทนที่ค่า 0 และ 1 เพื่อให้เป็นภาษาที่มนุษย์อ่านเข้าใจได้โดยง่าย คือ HIGH และ LOW

HIGH แปลว่าสูง หมายถึง 1 หรือ มีไฟ

LOW แปลว่าต่ำ หมายถึง 0 หรือ ไม่มีไฟ

การใช้งานก็แค่เขียน ชื่อขา และ สถานะที่ต้องการให้เป็น ลงไปในวงเล็บเท่านั้นเอง ง่ายมาก

digitalWrite(ชื่อขา,สถานะ);

digitalWrite(8,HIGH);

ตัวอย่างการสั่งให้ขา 8 มีสถานะเป็น 1 หรือเป็นการสั่งให้ขา 8 มีไฟออกมาเท่าแรงดันไฟเลี้ยงไอซี

****

digitalWrite(8,LOW);

ตัวอย่างการสั่งให้ขา 8 มีสถานะเป็น 0 หรือเป็นการสั่งให้ขา 8 ไม่มีไฟ

****

```
digitalWrite | Arduino 1.8.5
File Edit Sketch Tools Help
digitalWrite §

void setup() {
 pinMode(8,OUTPUT);
}

void loop() {
   digitalWrite(8,HIGH);
   delay(500);
   digitalWrite(8,LOW);
   delay(500);
}

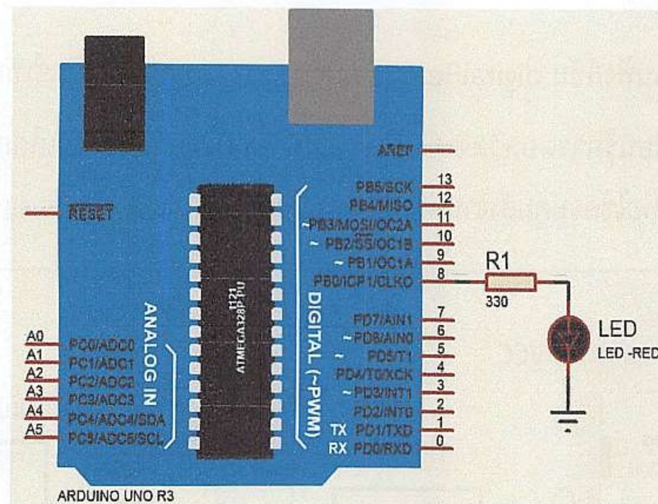
Done Saving.
```

โปรแกรมตัวอย่างการสั่งงานด้วยฟังก์ชัน digitalWrite()

ตัวอย่างการเขียนสั่งให้ขา 8 เป็นเอาต์พุต และสั่งให้มีสถานะเป็น HIGH หมายถึงมีไฟ เมื่อต่อหลอด LED เข้าไปที่ขา 8 หลอดไฟจะติดสว่างขึ้น และหน่วงเวลาไว้ครึ่งวินาที แล้วสั่งให้ขา 8 เป็น LOW หมายถึงไม่มีไฟ ทำให้ LED ดับลง และหน่วงเวลาไว้อีกครึ่งวินาที การทำงานของโปรแกรมนี้จึงมีผลทำให้

หลอด LED ที่ต่ออยู่กับขา 8 นั้น ติด และ ดับ สลับกันครั้งละครึ่งวินาที กลายเป็นไฟกระพริบนั่นเอง และ LED จะกระพริบตลอดเวลาที่ยังมีไฟเลี้ยงวงจร

แต่สิ่งสำคัญที่จะลืมไม่ได้ก็คือการที่จะสั่งงานให้ขาใดใด ส่งค่าดิจิทัลออกมา เป็น HIGH หรือ LOW ด้วยฟังก์ชัน digitalWrite() นั้น จะต้องมีการประกาศไว้แล้วว่าต้องการที่จะใช้งานขานั้นให้ทำงานเป็นเอาต์พุต ด้วยฟังก์ชัน pinMode(ขาที่จะใช้,OUTPUT);



ฟังก์ชัน digitalRead()

digitalRead();

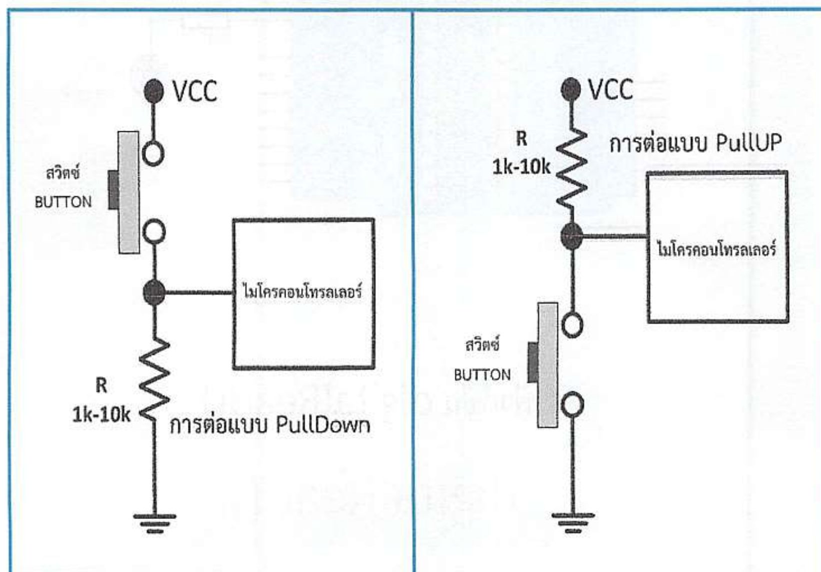
digitalRead(); เป็นฟังก์ชันสำเร็จรูปที่ใช้สำหรับอ่านข้อมูลดิจิทัลจากขาต่างๆ สามารถอ่านค่าได้จากทั้งขาที่ทำงานเป็น อินพุต และขาที่ทำงานเป็น เอาต์พุตก็ได้ ส่วนใหญ่ใช้สำหรับการอ่านค่าอินพุตจากการกดสวิตซ์ button เพื่อสั่งให้ทำงานต่างๆตามต้องการเช่น ปิด-เปิดไฟ สั่งมอเตอร์หมุน-หยุดหมุน การตั้งเวลา หรือ การเลือกโหมดการทำงานตามฟังก์ชันต่างๆโดยการใช้สวิตซ์แบบดิจิทัล การใช้งานนั้นเพียงแค่ใส่ชื่อขาลงในวงเล็บ ฟังก์ชันจะนำค่าข้อมูลไปตรวจสอบเองว่าขานั้นอยู่ในสถานะ HIGH หรือ LOW เพราะเป็นการตรวจสอบแบบดิจิทัลค่าของข้อมูลที่ได้จึงมีแค่สองสถานะคือ **HIGH มีไฟ** หรือ **LOW ไม่มีไฟ** เท่านั้น ส่วนการนำค่าที่ตรวจสอบได้ไปใช้งานนั้น จะต้องใช้ร่วมกับคำสั่งควบคุมการทำงาน เช่นคำสั่ง **if** จึงจะนำค่าข้อมูลที่อ่านได้ไปใช้งานให้เกิดประโยชน์ได้ และการใช้งานฟังก์ชัน **djgitalRead()** ไม่จำเป็นต้อง pinMode() ในการกำหนดว่าขานั้นจะใช้เป็นอินพุต หรือ เอาต์พุต เพราะว่าฟังก์ชัน **digitalRead()** นั้นสามารถใช้อ่านค่าดิจิทัลได้ทั้งหมด ไม่ว่าขานั้นจะทำงานเป็นอินพุต หรือ เอาต์พุตก็ตาม สามารถเรียกใช้งาน **digitalRread()** ได้ทันที

digitalRead(ชื่อขา);

digitalRead(7);

ตัวอย่างการใช้งานฟังก์ชัน digitalRead() เพียงแค่ใส่ชื่อขาที่ต้องการอ่านค่า ลงในวงเล็บ จากตัวอย่างนี้เป็นการอ่านค่าดิจิทัลจากขา7 เป็นการตรวจสอบว่าขา 7 นั้นมีไฟหรือไม่มี

การใช้งานฟังก์ชัน digitalRead() ในการตรวจสอบการกดสวิตซ์เพื่อนำไปสั่งให้ทำงานตามที่เรากำหนดนั้นจะต้องเรียนรู้การต่อสวิตซ์เข้ากับขาไมโครคอนโทรลเลอร์กันเสียก่อน การต่อสวิตซ์เข้ากับขาไมโครคอนโทรเลอร์เพื่อใช้ควบคุมสั่งงานต่างๆนั้น จะมีหลักใหญ่ๆอยู่ด้วยกันสองแบบ

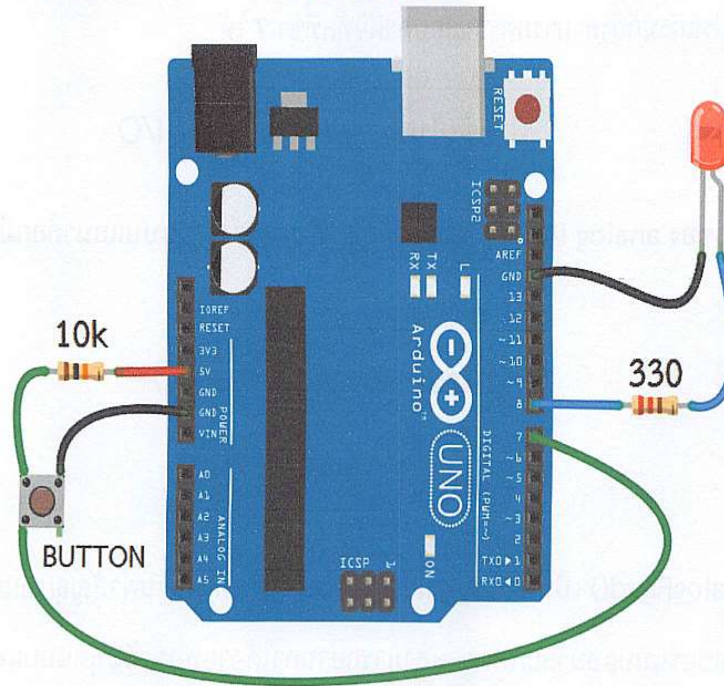


การต่อแบบพูลดาวน์ **Pull Down** เป็นการต่อให้ขานั้นมีสถานะเป็น LOW หรือเป็น 0 ไม่มีไฟ โดยการต่อ R ค่าประมาณ 1k-10k ต่อลงกราวด์ไป และจะใช้สวิตซ์ในการต่อรับไฟ 5V มาจากไฟเลี้ยงไอซี เมื่อมีการกดสวิตซ์จะทำให้มีไฟเข้ามาที่ขานั้น ทำให้มีค่าเป็น HIGH หรือมีไฟ จึงเขียนโปรแกรมรับค่า HIGH หรือ 1 นี้ไปสั่งงานแต่การต่อแบบนี้มีความเสี่ยงต่อสัญญาณรับกวนที่อาจเกิดขึ้นได้ สัญญาณรบกวนต่างๆนั้นจะมีค่าเป็นแรงดันไฟบวกทั้งสิ้น บางครั้งอาจจะทำให้ไมโครคอนโทรลเลอร์ประมวลผลผิดพลาด ว่าแรงดันไฟที่เกิดจากสัญญาณรบกวนที่เข้ามานั้น มาจากการกดสวิตซ์ อาจจะสั่งทำงานโดยที่เราไม่ได้กดสวิตซ์ก็ได้ จากข้อเสียตรงนี้การต่อแบบนี้ในการใช้งานจริง จึงไม่เป็นที่นิยมมากนัก

การต่อแบบพูลอัพ **Pull UP** เป็นการต่อให้ขานั้นมีสถานะเป็น HIGH หรือมีไฟไว้ก่อน โดยการใช้ R ค่าประมาณ 1k-10k รับแรงดันมาจากไฟ 3.3V - 5V ส่วนสวิตซ์นั้นจะต่อกับกราวด์ของวงจร เมื่อมีการกดสวิตซ์ก็จะทำให้ขานั้นเป็น LOW หรือไม่มีไฟ เพราะถูกดึงลงกราวด์ จึงเขียนโปรแกรมรับค่า LOW หรือ 0 นี้ไปสั่งงาน ไม่มีสัญญาณรบกวนใดที่จะมีค่าเป็น 0V การต่อใช้งานแบบพูลอัพ (Pull UP) นี้จึงเป็นที่นิยมนำไปใช้งานใน

เครื่องใช้อิเล็กทรอนิกส์ทั่วไปอย่างกว้างขวาง และในหนังสือเล่มนี้ เราก็จะใช้การต่อวงจรแบบนี้ทุกโครงงานที่มีการใช้สวิตซ์ เพื่อความถูกต้องและแม่นยำในการทำงาน

การต่อวงจรทดลอง



sketch_feb09a | Arduino 1.8.5

File Edit Sketch Tools Help

sketch_feb09a

```
void setup() {
  pinMode(8,OUTPUT);
}
void loop() {
  if(digitalRead(7)==0) digitalWrite(8,HIGH);
  else  digitalWrite(8,LOW);
}
```

โค้ดที่ใช้ในการทดลอง

จากตัวอย่างในรูป เป็นการเขียนโปรแกรมรับค่าดิจิทัลจากขา 7 โดยการต่อ R pull up ค่า10k ไว้กับไฟบวก 5 โวลต์เพื่อให้ขา 7 มีสถานะเป็น HIGH ไว้ก่อน หากมีการกดสวิตซ์จะทำให้ขา 7 ถูกต่อลงกราวด์ทำให้มีค่าเป็น

LOW หรือ 0 จึงเขียนคำสั่งให้ขา 8 มีไฟเพื่อขับหลอด LED และหากปล่อยสวิตซ์ออกทำให้ขา 7 มีค่าเป็น HIGH อย่างเดิม จึงเขียนคำสั่งให้ขา 8 ส่งค่า LOW ออกมาคือไม่มีไฟ ทำให้หลอด LED ดับลงทันที นี่เป็นตัวอย่างการใช้งานฟังก์ชัน **digitalRead()** อ่านค่าจากการกดสวิตซ์ ในเบื้องต้น

และหากสังเกตในฟังก์ชัน **setup** จะพบว่าไม่มีการประกาศ **pinMode()** เอาไว้เลยว่าจะใช้ขา 7 เป็นอินพุต แต่โปรแกรมก็ยังสามารถตรวจสอบค่าดิจิทัลที่ขา 7 ได้

ฟังก์ชันในหมวดของ analog I/O

ฟังก์ชันในหมวดของ analog I/O เป็นฟังก์ชันที่เกี่ยวกับการสั่งงานแบบแอนาล็อกมีอยู่ด้วยกัน 3 ฟังก์ชัน

ฟังก์ชัน AnalogRead()

analogRead( );

AnalogRead() เป็นฟังก์ชันสำเร็จรูปที่ใช้สำหรับการอ่านค่าสัญญาณแอนาล็อกจากเซ็นเซอร์ต่างๆ ที่เป็นเซ็นเซอร์แบบแอนาล็อก โดยจะสามารถอ่านค่าได้จากพอร์ตอินพุตแบบแอนาล็อก A0-A5 เท่านั้น โดยการแปลงสัญญาณแอนาล็อกให้เป็นดิจิทัลด้วยวงจรภายในของไอซีไมโครคอนโทรลเลอร์ เรียกว่า ADC (Analog to Digital Converter) สัญญาณที่รับเข้ามานั้นจะต้องมีแรงดันไฟไม่เกินค่าแรงดันไฟเลี้ยงของไอซี ในบอร์ด UNO R 3 นั้นจะอยู่ที่ 0 -5V แต่ในบอร์ดรุ่นอื่นๆ ต้องดูจากรายละเอียดของบอร์ดรุ่นนั้น บางรุ่นอาจใช้ไฟเลี้ยงเพียงแค่ 3.3V

หลักการทำงานของ ADC นั้นจะแปลงค่าแรงดันไฟที่รับเข้ามา 0-5V และแรงดันไฟที่รับเข้ามานี้เป็นสัญญาณแอนาล็อก แรงดันไฟจะถูกแปลงให้ค่าดิจิทัลด้วยการเฉลี่ยแบ่งระดับแรงดันนั้น ให้เป็นค่าตัวเลข 0-1023 โดย 0 หมายถึงค่าต่ำสุดและ 1023 เป็นค่าสูงสุด หากจะมองให้เข้าใจง่ายๆคือค่าตัวเลข 0 นั้นคือแรงดันไฟ 0V และค่าตัวเลข 1023 นั้นก็คือแรงดันไฟสูงสุดที่ใช้อ้างอิง ในบอร์ด UNO ปกติแรงดันไฟที่ใช้ในการอ้างอิงจะอยู่ที่ 5V แต่ต้องเข้าใจด้วยว่าแรงดันไฟสูงสุดที่ใช้ในการอ้างอิงนี้ เราสามารถเปลี่ยนแปลงใช้ค่าแรงดันที่ต่ำกว่า 5V ได้เพื่อความละเอียดในการวัด เราเรียกแรงดันไฟที่ใช้ในการอ้างอิงนี้ว่า Reference และเราจะใช้ค่าตัวเลขนี้เพื่อไปเข้าสูตรในการคำนวณต่างๆ แล้วแต่การใช้งานว่ารับสัญญาณนั้น มาจากเซนเซอร์ชนิดใด และ ในการใช้งานจริงจะต้องสร้างตัวแปรขึ้นมาเพื่อเก็บค่าตัวเลขนี้

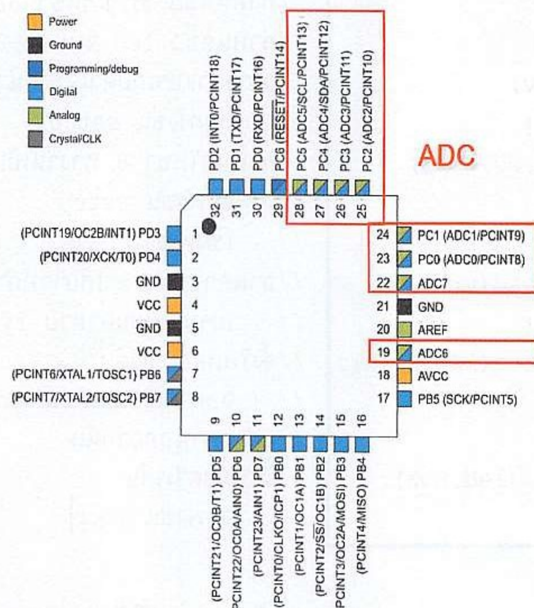
หากจะกล่าวถึง ADC ในไมโครคอนโทรลเลอร์แล้วต้องเข้าใจก่อนว่า ADC นั้นเป็นวงจร หรือ โมดูลพิเศษที่มีเพิ่มเข้ามา ในเฉพาะไมโครคอนโทรลเลอร์บางตัวเท่านั้น แต่ไมโครคอนโทรลเลอร์ในปัจจุบันส่วนใหญ่

จะมีมาให้ แทบทั้งหมด ยกเว้นตัวเล็กๆ ราคาถูกๆ บางตัวอาจจะไม่มี ADC มาให้ หรือบางตัวอาจจะมีมาให้เพียงช่องเดียว หรือ สองช่องเท่านั้น หากต้องการทราบว่าไอซีตัวไหนมี ADC หรือไม่มี หรือไอซีตัวนั้นมี ADC มาให้เราใช้งานได้กี่ช่อง ให้ดูจาก datasheet ของไอซีเบอร์นั้น สามารถดาวน์โหลดได้ฟรีจากเว็บไซต์ของผู้ผลิต กรณีนี้บอร์ดที่เราใช้เป็นบอร์ด UNO R3 SMD ใช้ไอซีเบอร์ ATMEGA328P เป็นไอซี SMD มีขาทั้งหมด 32 ขา มีขาใช้งาน ADC อยู่ 8 ขาแต่ในบอร์ด UNO R3 นั้นต่อออกมาให้ใช้งานเพียง 6 ขาเท่านั้น คือ ADC0 - ADC5 และพิมพ์ชื่อติดมาบนบอร์ด Arduino UNO R3 ว่า A0 - A5 ให้เข้าใจได้ทันทีว่าเป็น port ที่ใช้สำหรับการรับค่าข้อมูลที่เป็นข้อมูลแบบแอนาล็อก



ในการใช้ฟังก์ชัน analogRead(); จึงต้องต่ออุปกรณ์เข้ากับช่อง A0 - A5 เท่านั้น

แต่ในการใช้งานแบบดิจิทัล หรือ ใช้ฟังก์ชัน digitalRead() หรือ digitalWrite() นั้น เรายังสามารถใช้งานขา A0 - A5 เพื่อรับ หรือ ส่งสัญญาณแบบดิจิทัลก็ได้

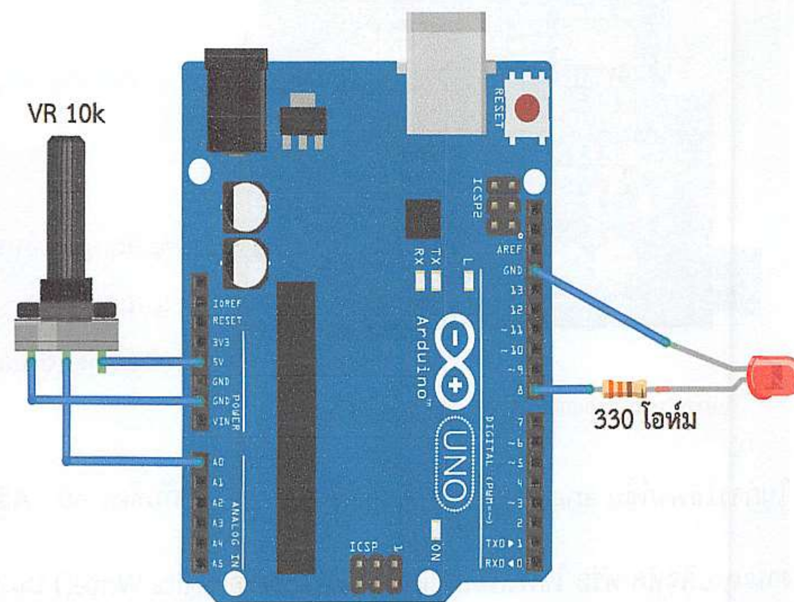


ภาพจาก datasheet แสดงขา ADC

ข้อมูลจากดาต้าชีท

Peripheral Features

- Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
- One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture
- Real Time Counter with Separate Oscillator
- Six PWM Channels
- 8-channel 10-bit ADC in TQFP and QFN/MLF package
  - Temperature Measurement
- 6-channel 10-bit ADC in PDIP Package
  - Temperature Measurement
- Two Master/Slave SPI Serial Interface
- One Programmable Serial USART
- One Byte-oriented 2-wire Serial Interface (Philips I^{2}C compatible)
- Programmable Watchdog Timer with Separate On-chip Oscillator




```
#define vr A0                    //กำหนดชื่อ vr แทนขา A0
#define led 8                    //กำหนดชื่อ led แทนขา 8
unsigned int v;                  //สร้างตัวแปรเก็บค่าเลขจำนวนเต็มชื่อ v
void setup() {                   // { เริ่มฟังชั่น setup
  pinMode(led,OUTPUT);           //กำหนดให้ขา 8 ทำงานเป็นเอาต์พุต
}                                // } จบฟังชั่น setup
void loop() {                    // { เริ่มฟังชั่น loop
  v = analogRead(vr);            //อ่านค่าจาก vr เก็บไว้ที่ตัวแปร v
  if(v > 512){                   // { เปรียบเทียบค่าด้วย if ถ้าหาก v มีค่ามากกว่า 512
    digitalWrite(led,HIGH);      //สั่งให้หลอดไฟติด
  }                              // { จบการเปรียบเทียบ  if
  else                           // else นอกจากนั้น
  digitalWrite(led,LOW);         //ให้หลอดไฟดับ
}                                // } จบฟังชั่น loop
```

ภาพตัวอย่างโปรแกรมที่ใช้งานจริง

โปรแกรมนี้เป็นการทดลองการใช้ฟังก์ชัน **analogRead()** ในการอ่านค่าจากอุปกรณ์ที่เป็นแอนาล็อก วงจรตัวอย่าง จะใช้ VR (Variable Resistors) ค่า 10k โดยเราจะต่อ ขา 1 ของ VR เข้ากับไฟบวก 5V และต่อขา 2 เข้ากับขา A0 ของบอร์ด UNO R3 เพื่อจะอ่านค่าความเปลี่ยนแปลงของการปรับ VR ส่วนขา 3 ของ VR นั้นต่อลงกราวด์ และเราจะนำค่าที่อ่านได้จากการปรับ VR นั้นมาสั่งงานให้หลอด LED ติด หรือดับ โดยการต่อ LED ไว้ที่ขา 8 ของบอร์ด UNO R3 หลักการทำงานของวงจรนี้คือเมื่อเราหมุนปรับ VR ไปเกินครึ่งหนึ่งแล้ว หลอด LED จะติดสว่างขึ้นมา และหากหมุนปรับลดลงมาต่ำกว่าครึ่ง หลอด LED ก็จะดับลง

เมื่อเรากำหนดการทำงานของวงจรไว้แล้ว ก็เหลือแต่การเขียนโปรแกรมให้วงจรทำงานตามที่เราต้องการ

#define vr A0 //เป็นการตั้งชื่อขึ้นมาเองว่า **vr** นั้นให้หมายถึงขา A0

#define led 8 //เป็นการตั้งชื่อขึ้นมาเองว่า **led** นั้นให้หมายถึงขา 8

ในการกำหนดชื่อแทนชื่อของขาขึ้นมาเองนั้น จะมีประโยชน์มากหาก โปรแกรมของเรามีขนาดใหญ่ เมื่อเขียนๆไปหลายร้อยบรรทัด เราอาจจะจำไม่ได้และสับสนเอง ว่าขาบางขานั้นเราได้ต่อวงจรไว้กับอะไรดังนั้น การกำหนดชื่อแทนขึ้นมา เพื่อให้สอดคล้องกับงานที่ทำนั้น จะทำให้เราเขียนโปรแกรมได้ง่ายขึ้นจากตัวอย่างนี้ เราได้ต่อ VR ไว้ที่ขา A0 และต่อหลอด LED ไว้ที่ขา 8 คำสั่ง **#define** จึงถูกนำมาใช้ให้เป็นประโยชน์เพื่อการนี้

unsigned int v; คือการสร้างตัวแปรชนิดเลขจำนวนเต็มเพื่อเก็บค่าตัวเลขที่อ่านได้จากขา A0

ในฟังก์ชันของการตั้งค่า **setup()** เป็นการกำหนดว่าจะใช้ขาใดเป็นอินพุต หรือเอาต์พุต

การใช้งานฟังก์ชัน **analogRead();** นั้นไม่จำเป็นต้องประกาศ **pinMode()** ว่าจะใช้งานเป็นอินพุต

pinMode(led,OUTPUT); กำหนดให้ขา 8 ทำงานเป็นเอาต์พุต เพื่อขับหลอด LED

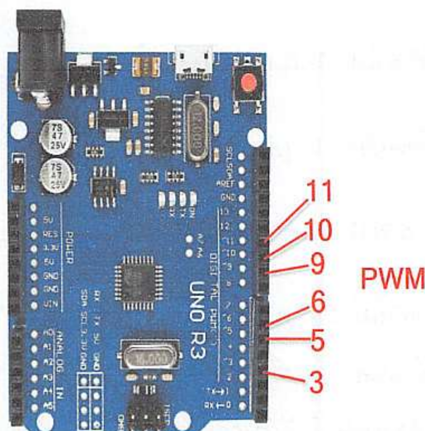
ต่อจากนั้นโปรแกรมจะเข้าทำงานในฟังก์ชัน **loop** และจะวนทำงานอยู่อย่างนั้นตลอดไป เราได้ทราบกันแล้วว่าค่าของข้อมูลที่อ่านได้จาก ADC นั้นจะมีค่า 0 – 1023 ดังนั้นค่าครึ่งหนึ่งของ 1024 ก็คือ 512 หากเราต้องการให้ปรับ VR ไปเกินครึ่งแล้วให้หลอด **led** ติดสว่างขึ้นมาจึงต้องใช้คำสั่งในการตรวจสอบเงื่อนไขโดยคำสั่ง **if** ว่าถ้าหากค่าเกิน 512 ไปแล้ว ก็ให้สั่งงานขา 8 จ่ายไฟออกมาให้หลอด led ติดด้วยฟังก์ชัน **digitalWrite();** นั่นเอง

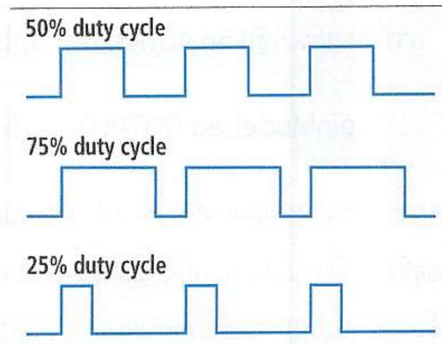
ฟังก์ชัน analogWrite()

analogWrite( );

analogWrite(); เป็นฟังก์ชันสำเร็จรูป ในการสร้างสัญญาณพัลส์ หรือ PWM (Pulse Width Modulation) โดยสามารถปรับค่าความกว้างของพัลส์ได้ ที่เรียกว่าดิวตี้ไซเคิล duty cycle เพื่อนำไปปรับความสว่างของหลอด LED ปรับความเร็วมอเตอร์ต่างๆโดยการใส่ชื่อขาและค่า ดิวตี้ไซเคิล ลงในวงเล็บ เท่านั้นเอง ในบอร์ด UNO R3 ขาที่ใช้งาน PWM ได้มีทั้งหมด 6 ขาด้วยกันคือขา 3,5,6,9,10 และ 11 และฟังก์ชันนี้จะมีความถี่คงที่ แต่ละขานั้นจะส่งความถี่ที่ไม่เท่ากัน สามารถสั่งให้ส่งสัญญาณออกได้พร้อมกัน ได้ทั้งหมด 6 ขา

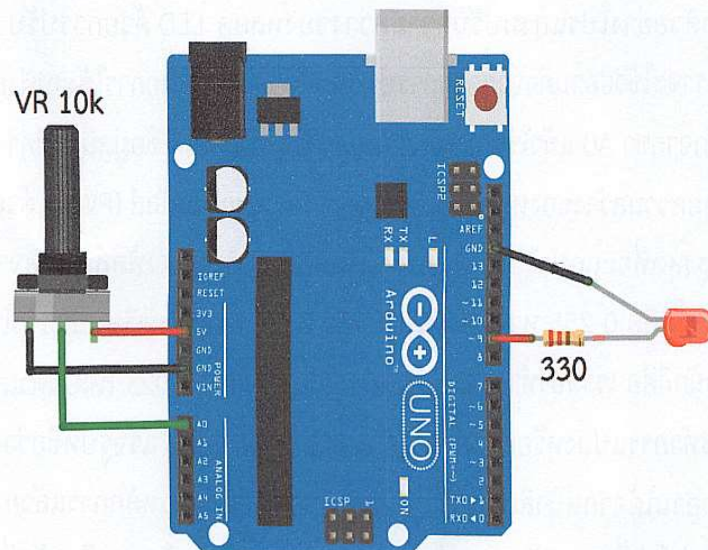
ตำแหน่งขา	ความถี่
3	490 Hz
5	980 Hz
6	980 Hz
9	490 Hz
10	490 Hz
11	490 Hz





การปรับเปลี่ยนค่า duty cycle นั้นในการใช้ฟังก์ชัน **analogWrite()** จะมีค่าตั้งแต่ 0-255 ซึ่งค่า 255 เป็นค่าสูงสุดหมายถึง 100% หากต้องการปรับเป็นค่า % ต้องใช้วิธีการเรียกใช้ฟังก์ชันอีกฟังก์ชันหนึ่งขึ้นมาเพื่อแบ่งค่าตัวเลข 0-255 ให้เป็นค่าตัวเลข 0-100 **%** นั่นคือฟังก์ชัน **map()** ให้ดูในบทของฟังก์ชัน **map()**

การเรียกใช้งานขา ฟังก์ชัน **analogWrite()** นั้นไม่จำเป็นต้องเรียกใช้ฟังก์ชัน **pinMode()** ว่าจะใช้งานเป็นเอาต์พุต เพราะฟังก์ชัน **analogRead()** และ **analogWrite()** จะกำหนดให้เอง โดยอัตโนมัติ




```
#define pwm 9                          //กำหนดตั้งชื่อ pwm ขึ้นมาแทนขา 9
#define vr A0                          //กำหนดตั้งชื่อ vr ขึ้นมาแทนขา A0
unsigned int v;                        //สร้างตัวแปรชนิดเลขจำนวนเต็ม ชื่อ v เพื่อเก็บค่าการปรับ VR
unsigned int duty;                     //สร้างตัวแปร duty เก็บค่าการแปลงค่าจาก 0-1024 ให้เป็น 0-255
void setup(){                          //เริ่มต้นฟังชั่น setup
                                       //*ทดลอง โดยไม่ต้องเขียนคำสั่ง กำหนดขาในการใช้งาน
}                                      //จบการทำงานฟังชั่น setup
void loop(){                           //เริ่มฟังชั่น loop()
  v = analogRead(vr);                  //อ่านค่าอนาล็อกที่ขา A0 มาเก็บค่าในตัวแปร v
  duty = map(v,0,1023,0,255);          //ใช้ฟังชั่น map() เพื่อแปลงค่า 0-1023 ให้เป็น 0-255
  analogWrite(pwm,duty);               //ส่งสัญญาณ PWM ออกที่ขา 9 ด้วยค่าที่อ่านได้จากการปรับ VR
}                                      //จบฟังชั่น loop
```

led | Arduino 1.8.5

File Edit Sketch Tools Help

led

```
void setup() {

}
void loop() {
  int vr = analogRead(A0);
  int dim = map(vr,0,1023,0,255);
   analogWrite(9,dim);
}
```

Done uploading.

ตัวอย่างโปรแกรมปรับความสว่างของหลอด LED อย่างง่าย

จากตัวอย่างโปรแกรมปรับความสว่างของหลอด LED ด้วยการปรับ VR หลักการทำงานของโปรแกรมนี้คือ เราจะใช้วิธีอ่านค่าจากจากการปรับ VR ด้วย ADC โดยการใช้งานฟังก์ชัน **analogRead()** เพื่ออ่านค่าแอนาล็อกจากขา A0 แล้วใช้ตัวแปร vr เก็บค่าข้อมูลที่อ่านได้ ข้อมูลนี้จะมีค่า 0-1023 แต่เราต้องการนำข้อมูลนี้ไปควบคุมความสว่างของหลอด LED โดยการส่งสัญญาณพัลส์ (PWM) ด้วยฟังก์ชัน **analogWrite()** แต่ว่าค่าตัวเลขสูงสุดที่จะถูกนำไปใช้เป็นค่าดิวตี้ไซเคิลของสัญญาณพัลส์ ที่เกิดจากการทำงานของฟังก์ชัน **analogWrite()** นั้นคือ 0-255 หากพูดเป็นภาษาที่ชาวบ้านทั่วไปฟังรู้เรื่องและเข้าใจได้ง่ายโดยไม่ต้องใช้ศัพท์ทางวิชาการมากนักก็คือ เราต้องทำการแปลงค่าช่วงตัวเลข จาก 0-1023 ให้เป็นตัวเลขที่อยู่ในช่วง 0-255 ก่อน และฟังก์ชันที่จะทำการแปลงหรือแบ่งค่าช่วงตัวเลขนี้ก็คือฟังก์ชันสำเร็จรูปที่ชื่อว่า **map()** เรื่องของฟังก์ชัน **map()** สามารถอ่านได้จากหนังสือเล่มนี้ในบทต่อๆไป เมื่อเข้าใจในหลักการแล้วการที่จะนำไปประยุกต์ใช้ก็ไม่ใช่เรื่องยากอีกต่อไป ซึ่งการปรับความเร็วของมอเตอร์ DC ก็ใช้หลักการเดียวกันนี่เอง หากแต่ว่า ถ้ามอเตอร์นั้นใช้แรงดันไฟสูงกว่า 5V ซึ่งสูงกว่าแรงดันไฟเลี้ยงของบอร์ด จะต้องมีการเพิ่มวงจรป้องกันความเสียหายที่อาจะเกิดขึ้นได้ ด้วยอุปกรณ์ไอโซเลเตอร์ (isolators) เช่นออปโต้ ในเนื้อหาเบื้องต้นนี้ผู้เขียนจะยังไม่กล่าวถึงเรื่องของระบบขับกำลัง ที่ใช้ในการขับมอเตอร์แบบต่าง

ฟังก์ชัน analogReference()

analogReference( );

analogReference() เป็นฟังก์ชันที่ใช้สำหรับกำหนดค่าแรงดันไฟ เพื่อใช้ในการอ้างอิงในการอ่านค่าจาก ADC ด้วยฟังก์ชัน **analogRead()** โดยค่าแรงดันสูงสุดที่อ่านค่าได้จาก ADC จากฟังก์ชัน analogRead() นั้นจะมีค่าเท่ากับแรงดันอ้างอิง ที่มาจากการทำงานของฟังก์ชัน analogReference() โดยฟังก์ชัน analogReference() นี้หากเป็นบอร์ดรุ่น UNO R3 นั้นจะมีการทำงานทั้งหมด 3 โหมด คือ DEFAULT, INTERNAL และ EXTERNAL หากมีการเรียกเปิดใช้งานฟังก์ชัน analogReference() จะส่งผลให้ analogRead() อ่านค่าได้สูงสุดแค่เท่ากับแรงดันไฟอ้างอิงเท่านั้น ยกตัวอย่างเช่น หากกำหนดแรงดันไฟอ้างอิงไว้ที่ 3V เมื่อเราทำการป้อนแรงดันไฟ 3V เข้าที่ขา A0 และวัดค่าออกมาด้วย ฟังก์ชันanalogRead(A0); จะอ่านค่าจาก ADC ได้ค่า 1023 ซึ่งเป็นค่าสูงสุดแล้ว และถึงแม้ว่าจะจ่ายไฟเข้าไปมากกว่า 3V หรือ 4V หรือ 5V ก็จะยังอ่านค่าได้ 1023 หมายความว่า หากตั้งแรงดันอ้างอิงไว้ที่ 3V แล้ว ฟังก์ชัน analogRead() จะอ่านค่าแรงดันไม่ได้ไม่เกิน 3V แต่ความละเอียดในการวัดแรงดัน 0-3V จะมีความละเอียดมากขึ้น นี่คือความหมายของการใช้งานแรงดันไฟอ้างอิง analogReference()

แรงดันไฟอ้างอิงนี้จะอยู่ที่ขา AREF สามารถใช้ Volt Meter วัดดูได้ และเมื่อเรียกใช้งานฟังก์ชัน analogReference() แล้ว ค่าแรงดันอ้างอิงนี้จะถูกนำไปใช้เป็นค่าสูงสุด ที่จะอ่านได้จากฟังก์ชัน analogRead()

ดังที่กล่าวมา ยกตัวอย่างเช่นโหมด DEFAULT เป็นการอ้างอิงแรงดันไฟที่ 5V นั่นหมายความว่าหากฟังก์ชัน analogRead() อ่านค่า ADC ออกมาได้ 1023 ซึ่งเป็นค่าสูงสุด แสดงว่าแรงดันไฟที่อ่านได้จากขานั้นมีแรงดันไฟ 5V นั้นเอง หรือ หากอ่านค่า ADC ออกมาได้ 512 แสดงว่าแรงดันไฟที่ขานั้นมี 2.5V

โหมดที่1. analogReference(DEFAULT);

DEFAULT: เป็นค่าแรงดันไฟอ้างอิงตามปกติ จะเท่ากับแรงดันไฟเลี้ยงไอซี สำหรับบอร์ด UNO จะมีอยู่สองรุ่นคือรุ่นที่ใช้ไฟ 5V แรงดันไฟอ้างอิงก็จะอยู่ที่ 5V และหากเป็นบอร์ดรุ่นที่ใช้ไฟเลี้ยง 3.3V แรงดันไฟอ้างอิงก็จะอยู่ที่ 3.3V เท่ากับแรงดันไฟเลี้ยง และหากแรงดันไฟเลี้ยงตก เช่นจาก 5V เหลือเพียง 4.8 แรงดันไฟอ้างอิงก็จะตกลงเหลือ 4.8V ด้วยเช่นกัน โดยเฉพาะการใช้ไฟเลี้ยงจากพอร์ต USB ซึ่งจ่ายกระแสได้น้อยมาก เมื่อโหลดกินกระแสเกินกว่าที่จะจ่ายได้ แรงดันไฟจะลดลง อาจจะไม่ได้แรงดันไฟ 5V ตามที่ต้องการ และเมื่อแรงดันไฟที่ใช้ในการอ้างอิงนั้นผิดเพี้ยนไปแล้ว ก็จะส่งผลให้ค่าที่อ่านได้นั้น ผิดเพี้ยนไปจากความเป็นจริงด้วย ทำให้ค่าต่างๆที่เราต้องการวัดนั้นผิดพลาดได้ อย่างเช่นการใช้เซ็นเซอร์ที่ใช้วัดอุณหภูมิแบบแอนาล็อก หากแรงดันไฟที่ใช้อ้างอิงนั้นผิดเพี้ยนไป ก็จะส่งผลให้ค่าอุณหภูมิที่วัดออกมานั้นผิดเพี้ยนไปด้วย ดังนั้นหากจะใช้โหมดนี้ควรใช้แรงดันไฟเลี้ยง 7V-12V เสียบเข้าที่ช่อง VIN จะได้แรงดันไฟที่นิ่งกว่า เพราะว่าในบอร์ด Arduino มีไอซีสำหรับลดแรงดันไฟและควบคุมแรงดันไฟให้คงที่ ใส่มาให้แล้ว

โหมดที่2. analogReference(INTERNAL);

INTERNAL: เป็นการเรียกใช้การอ้างอิงค่าแรงดันไฟ จากวงจรที่อยู่ภายในตัวไอซี โดยบอร์ด UNO ที่ใช้ไอซีเบอร์ ATmega168 หรือ ATmega328P นั้นจะมีค่าแรงดันไฟอ้างอิงอยู่ที่ 1.1V แต่หากเป็นไอซีเบอร์ ATmega8 จะมีแรงดันอ้างอิงอยู่ที่ 2.56V

แต่ถ้าหากใช้บอร์ดรุ่น Mega สามารถจะเลือกแรงดันไฟอ้างอิงจากภายในได้ทั้งสองค่า คือ 1.1V หรือ 2.56V ก็ได้ ดังนั้นโหมดการใช้งานแรงดันอ้างอิงจากภายในจะใช้คำว่า INTERNAL กับบอร์ดรุ่น Mega ไม่ได้ จะต้องใช้คำว่า INTERNAL1V1 เพื่อใช้แรงดัน 1.1V หรือ INTERNAL2V56 เพื่อใช้แรงดัน 2.56V และคำนี้จะใช้ในบอร์ด Mega เท่านั้น

โหมดที่3. analogReference(EXTERNAL);

EXTERNAL: เป็นโหมดที่ใช้แรงดันไฟอ้างอิงจากภายนอก หมายความว่าหากใช้งานโหมดนี้จะต้องต่อแรงดันไฟอ้างอิงจากภายนอกเข้ามาในบอร์ด โดยต่อเข้ามาที่ขา AREF และแรงดันไฟที่จะต่อเข้ามานั้นจะต้องอยู่ในช่วงตั้งแต่ 0 โวลต์ ถึง 5 โวลต์ เท่านั้น

ฟังก์ชันในหมวดของ Time

ฟังก์ชันในหมวดของ Time เป็นฟังก์ชันที่เกี่ยวกับ เวลา การนับเวลา และหน่วงเวลาต่างๆ เป็นการใช้งานไทม์เมอร์ ด้วยฟังก์ชันสำเร็จรูป มีอยู่ด้วยกัน 4 ฟังก์ชัน

ฟังก์ชัน millis()

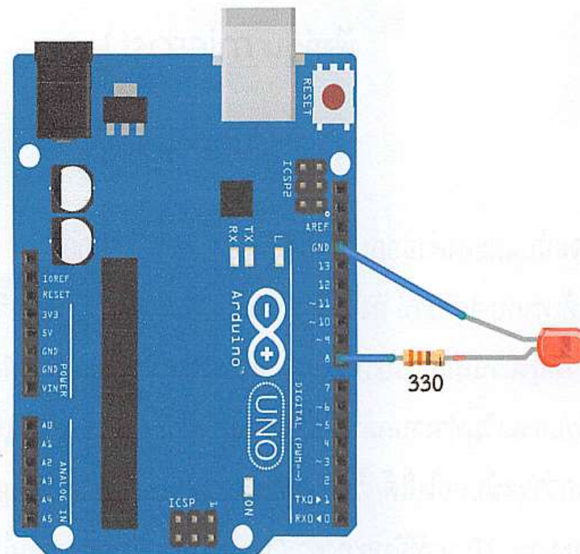
millis();

millis() เป็นฟังก์ชันสำเร็จรูป เกี่ยวกับการนับเวลาของไทม์เมอร์ภายในไอซีไมโครคอนโทรลเลอร์ที่อยู่ในบอร์ด Arduino โดยจะนับไปเรื่อยๆ ตั้งแต่เริ่มต้นการทำงาน และจะนับตลอดเวลาที่โปรแกรมทำงานอยู่ และจะกลับมานับเริ่มต้นใหม่ที่ 0 หลังจากนับไป ประมาณ 50 วัน หรือมีการหยุดจ่ายไฟ หรือ มีการรีเซ็ตการทำงาน และเมื่อเรียกใช้ฟังก์ชัน millis() มันจะส่งค่าการนับออกมามีหน่วยเป็นมิลลิวินาที การใช้งานนั้นจะต้องใช้ตัวแปรในการเก็บค่าข้อมูลที่นับเป็นชนิด unsigned long เนื่องจากการนับเพิ่มค่าตัวเลขขึ้นไปถึง 50 วันนั้นจะทำให้ได้จำนวนตัวเลขที่เยอะมาก หากจะเก็บค่าการนับจึงต้องใช้พื้นที่ในหน่วยความจำเยอะพอสมควร

เราสามารถนำมาประยุกต์ใช้งานสำหรับการนับของเราได้ การใช้ฟังก์ชันนี้มีข้อดีคือ เป็นการนับจากการทำงานของไทม์เมอร์ภายในตัวไอซี ไม่ใช่การหน่วงเวลาให้หยุดการทำงาน ดังนั้นการใช้งานการนับแบบนี้ ไมโครคอนโทรลเลอร์จึงสามารถทำงานอื่นๆ ไปพร้อมๆกันได้

```
#define led 8                          //กำหนดตั้งชื่อ led ขึ้นมาแทนขา 8
unsigned long sec;                     //สร้างตัวแปรเก็บเลขจำนวนเต็มแบบยาว ชื่อ sec
boolean ss = false;                    //สร้างตัวแปรชนิดชนิดบูลีน ชื่อ ss และกำหนดให้มีค่าเป็นเท็จ
void setup(){                          //เริ่มต้นฟังชั่น setup
  pinMode(led,OUTPUT);                 //กำหนดให้ขา 8 ทำงานในโหมดเอาต์พุต
}                                      //จบการทำงานฟังชั่น setup
void loop(){                           //เริ่มฟังชั่น loop()
  if(millis() > sec){                  //เปรียบเทียบ ถ้า millis() มีค่ามากกว่าตัวแปร sec
    sec = 1000+millis();               //ให้ sec มีค่ากับ 1000 บวกด้วยค่าของ millis()
    ss =! ss;                          //ให้กลับสถานะตัวแปร boolean
    if(ss == true)                     //เปรียบเทียบค่า ss หากมีค่าเป็นจริง
    digitalWrite(led,HIGH);            //สั่งให้หลอดไฟติด
    else                               //นอกนั้น
    digitalWrite(led,LOW);             //ให้หลอดไฟดับ
  }                                    //จบการทำงานเปรียบเทียบ
}                                      //จบฟังชั่น loop
```

โปรแกรมตัวอย่างการใช้งานฟังก์ชัน millis()



วงจรตัวอย่างใช้ LED ต่อเข้ากับขา 8 ของบอร์ด UNO R3 การทำงานของวงจรนี้ หลอด LED จะติดค้าง 1 วินาที และ ดับ 1 วินาที เป็นอย่างนี้สลับกันไป หรือจะประกาศตัวแปรแบบค่าคงที่ใช้งานเพื่อกำหนดเวลาในการกระพริบให้ช้าหรือเร็วก็ได้ จากตัวอย่างไฟกระพริบแบบไม่ต้องใช้ฟังก์ชัน delay() เลย

```
//ตัวอย่างไฟกระพริบที่ไม่ต้องใช้ฟังชั่น delay()
#define led 8
int led_state = LOW;
unsigned long p = 0;
const long interval = 500;
void setup() {
  pinMode(led,OUTPUT);
}

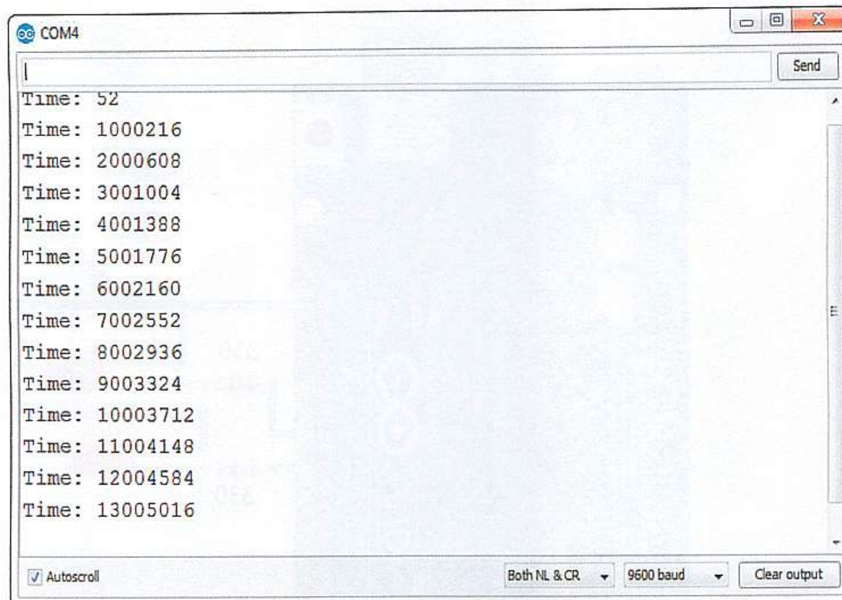
void loop() {
  unsigned long t = millis();
  if (t - p >= interval) {
    p = t;
    if (led_state == LOW) {
      led_state = HIGH;
    } else {
      led_state = LOW;
    }
    digitalWrite(led,led_state);
  }
}
```

ฟังก์ชัน micros()

micros();

ฟังก์ชันนี้จะนับและส่งค่าออกมาเป็นไมโครวินาที โดย 1,000 ไมโครวินาที จะได้ 1 มิลลิวินาที และ 1,000 มิลลิวินาที จะได้เท่ากับ 1 วินาที ดังนั้นการนับด้วยฟังก์ชันนี้นั้น ใน 1 วินาที จะนับได้ค่าตัวเลขประมาณ 1,000,000 ซึ่งหมายถึงหนึ่งล้านไมโครวินาทีนั่นเอง โดยหลักการทำงานของฟังก์ชัน micros() นั้น โปรแกรมจะทำการเริ่มนับตั้งแต่โปรแกรมเริ่มทำงานคล้ายกับฟังก์ชัน millis() และจะนับเพิ่มค่าขึ้นไปเรื่อยๆจนกว่าจะถึงค่าค่าหนึ่ง ที่เป็นค่าที่เกินกว่าจะนับต่อไปได้ เรียกว่าการ Overflow เป็นการนับตั้งแต่ค่าเริ่มต้นไปจนถึงค่าสูงสุดที่จะนับได้ จะใช้เวลาประมาณ 70 นาทีจึงจะถึงค่า Overflow แล้วจะวนกลับไปเริ่มนับที่ 0 ใหม่อีกครั้ง และนับต่อไปเรื่อยๆ และหากเราทดลองเขียนโปรแกรมให้ส่งค่าที่นับได้ให้แสดงผลออกมาดูทุกๆ 1 วินาที ก็จะพบว่าค่าที่ส่งออกมานั้นจะได้ตัวเลขอยู่ที่ประมาณหนึ่งล้าน หมายถึงในเวลา 1 วินาทีนั้น จะนับได้ประมาณหนึ่งล้านหน่วย ในการใช้งานฟังก์ชัน micros() นี้สามารถนำมาใช้งานสำหรับการนับเวลาที่ต้องการความละเอียดมากในระดับไมโครวินาที

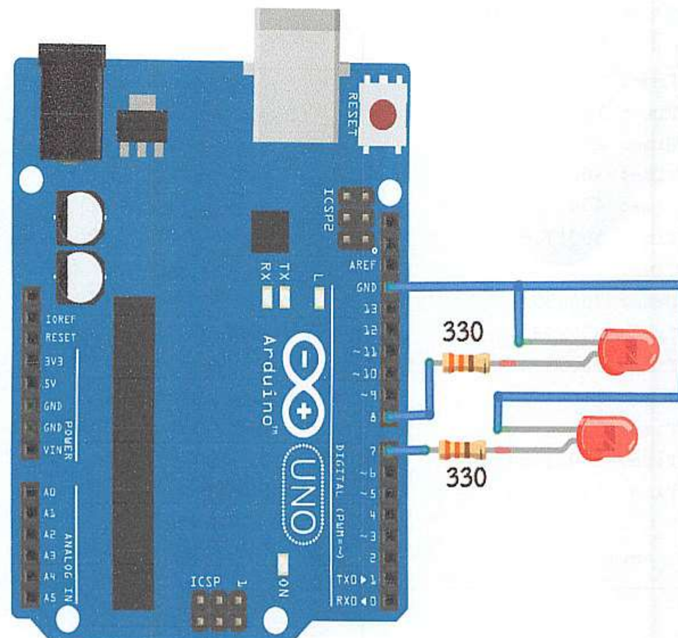
```
unsigned long time;
void setup(){
  Serial.begin(9600);
}
void loop(){
  Serial.print("Time: ");
  time = micros();
  Serial.println(time);
  delay(1000);
}
```



ฟังก์ชัน Delay()

delay();

Delay() เป็นฟังก์ชันที่ใช้ในการหน่วงเวลาการทำงาน โดยการใส่ค่าตัวเลขที่ต้องการหน่วงเวลาลงในวงเล็บ มีหน่วยเป็น มิลลิวินาที ฟังก์ชันนี้จะหน่วงเวลาโดยหยุดการทำงานทุกอย่างไว้ จนกว่าจะครบเวลาที่กำหนดจึงจะทำงานตามคำสั่งในบรรทัดต่อไป ฟังก์ชันนี้ถูกนำไปใช้เพื่อหน่วงเวลาในการทำไฟกระพริบ ของผู้เริ่มต้นศึกษาไมโครคอนโทรลเลอร์เกือบทั้งหมด ด้วยการใช้งานที่ง่ายกว่าการหน่วงเวลาด้วยฟังก์ชันอื่นๆ แต่ในการใช้งานจริงนั้นหากใช้งานบอร์ด Arduino ให้ทำงานที่หลากหลายที่ต้องการความรวดเร็วและแม่นยำนั้นควรหลีกเลี่ยงการใช้งานฟังก์ชัน delay() นี้เพราะหากใช้หน่วงเวลานานและบ่อยเกินไปนั้นจะส่งผลกระทบให้ไมโครคอนโทรลเลอร์เสียเวลา ในการหน่วงเวลานี้ จนทำให้การทำงานในการประมวลผลอย่างอื่นช้าและผิดพลาดได้ หากต้องการจับเวลาเพื่อกระทำการบางอย่างควรใช้ฟังก์ชันที่เป็นการอินเตอร์รัปต์จากไทม์เมอร์ เช่นฟังก์ชัน millis() หรือ micros() แทนจะดีกว่า เพราะหากใช้งานฟังก์ชัน delay() แล้วโปรแกรมจะวนทำงานอยู่ในลูปของการนับในฟังก์ชัน delay() จนกว่าจะถึงเวลาที่กำหนดไว้ในวงเล็บจึงจะออกไปทำงานอย่างอื่นได้เพราะความหมายของ delay() นั้นคือการ หน่วงเวลา




```
#define led1 7                      //กำหนดชื่อ led1 ขึ้นมาแทนขา 7
#define led2 8                      //กำหนดชื่อ led2 ขึ้นมาแทนขา 8
void setup() {
  pinMode(led1,OUTPUT);             //กำหนดให้ขา 7 ทำงานเป็นเอาต์พุต
  pinMode(led2,OUTPUT);             //กำหนดให้ขา 8 ทำงานเป็นเอาต์พุต
}
void loop() {
  digitalWrite(led1,HIGH);          //สั่งให้ขา 7 จ่ายไฟออกมาให้หลอด LED1 ติด
  digitalWrite(led2,LOW);           //กำหนดให้ขา 8 เป็น 0 ไม่มีไฟ ให้ LED2 ดับ
  delay(500);                       //ใช้ฟังก์ชั่น delay() หน่วงเวลาไว้ 500 มิลลิวินาที
  digitalWrite(led1,LOW);           //กำหนดให้ขา 7 เป็น 0 ไม่มีไฟ ให้ LED1 ดับ
  digitalWrite(led2,HIGH);          //กำหนดให้ขา 8 เป็น 1 มีไฟ ให้ LED2 ติด
  delay(500);                       //ใช้ฟังก์ชั่น delay() หน่วงเวลาไว้ 500 มิลลิวินาที
}
```

ตัวอย่างการใช้งานฟังก์ชัน delay()

ฟังก์ชัน delay() หรือการหน่วงเวลานั้นจึงเหมาะกับงานบางชนิด ที่ไม่มีการทำงานหลายๆอย่างในเวลาเดียวกัน อย่างเช่นการทดลองสร้างไฟกระพริบ หรือไฟวิ่ง ส่วนการทำงานที่ต้องการความรวดเร็วและแม่นยำ ที่เกี่ยวข้องกับชีวิต และทรัพย์สิน นั้นควรหลีกเลี่ยงการใช้ฟังก์ชัน delay()

ฟังก์ชัน delayMicroseconds()

delayMicroseconds();

delayMicroseconds() เป็นฟังก์ชันที่ใช้ในการหน่วงเวลาการทำงาน เหมือนกันกับฟังก์ชัน **delay()**แต่ต่างกันเพียงแค่หน่วยเวลาในการหน่วงนั้น มีหน่วยเป็นไมโครวินาที ใช้ในกรณีที่ต้องการหน่วงเวลาสั้นๆที่มีค่าเวลาต่ำกว่า 1 มิลลิวินาที โดยการใช้งานนั้นเหมือนกับฟังก์ชัน **delay()** ทุกประการ

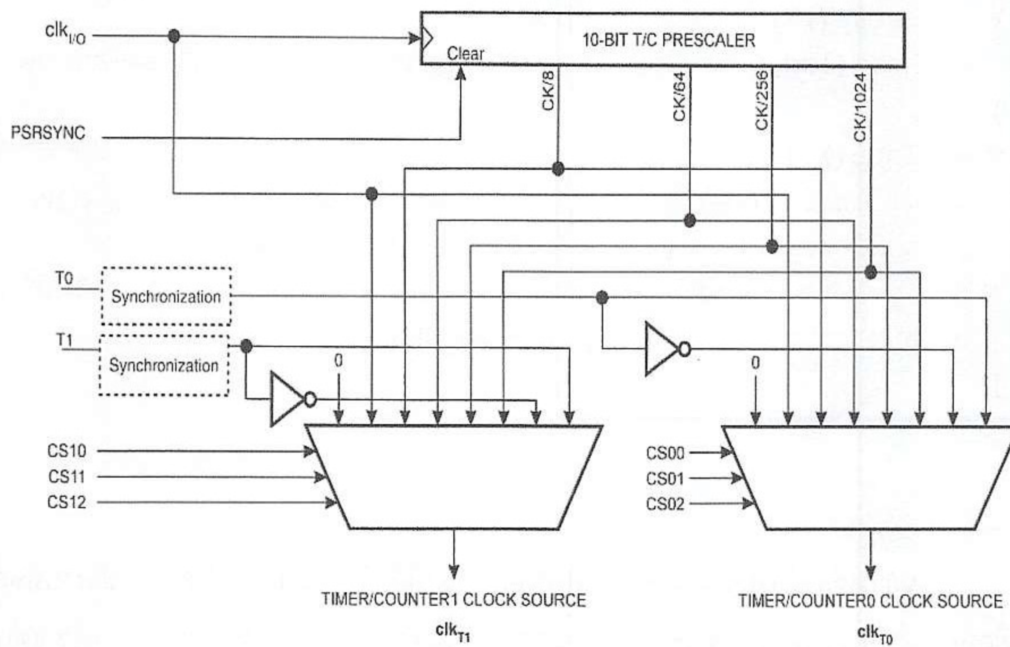
```
#define led 8                      //กำหนดชื่อ led ขึ้นมาแทนขา 8
void setup() {
  pinMode(led,OUTPUT);               //สั่งให้ขา 8 ทำงานในโหมดเอาต์พุต
}
void loop() {
  digitalWrite(led,HIGH);          //สั่งให้ขา 8 มีสถานะเป็น HIGH มีไฟ
  delayMicroseconds(500);          //หน่วงเวลา 500 ไมโครวินาที
  digitalWrite(led,LOW);           //สั่งให้ขา 8 มีสถานะเป็น LOW  ไม่มีไฟ
  delayMicroseconds(500);          //หน่วงเวลา 500 ไมโครวินาที
}
```

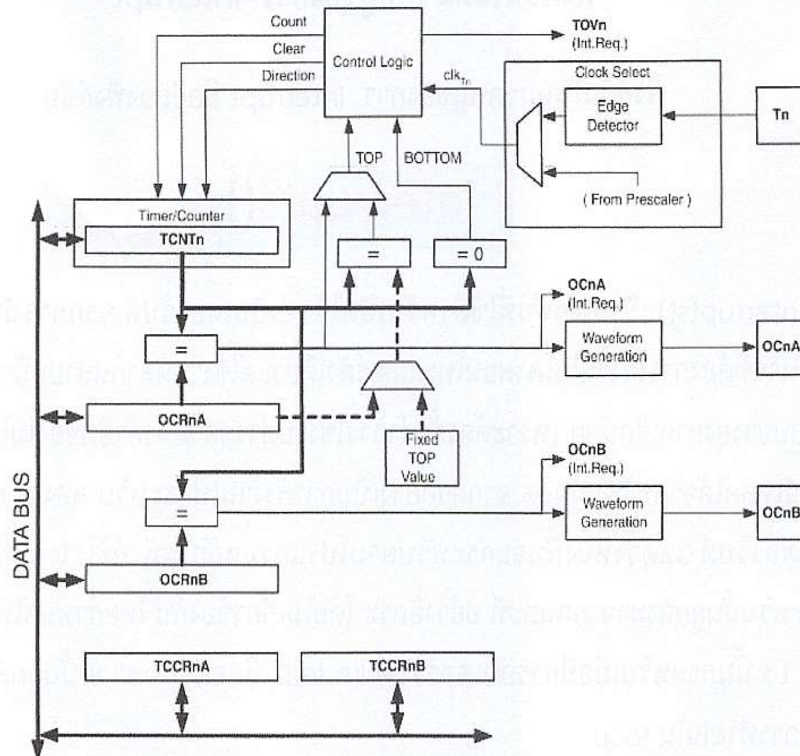
ในการเขียนโปรแกรมไมโครคอนโทรลเลอร์ สิ่งที่จำเป็นอีกอย่างหนึ่งคือการทำงานที่หลากหลายและยืดหยุ่น ด้วยไมโครคอนโทรลเลอร์นั้น สามารถทำงานหลายๆอย่างได้ ในเวลาอันรวดเร็ว แต่ในบางครั้งหากเราต้องการสั่งให้ทำงานบางอย่างที่เร่งด่วนก่อน หรือ ต้องการให้ไมโครคอนโทรลเลอร์หยุดทำงานหลักก่อนเพื่อไปทำงานที่เร่งด่วนหรืองานที่สำคัญกว่า เสร็จแล้วจึงให้กลับมาทำงานต่อจากเดิม เราเรียกสิ่งนี้ว่าการอินเตอร์รัปต์ (interrupt) หรือ การขัดจังหวะ หรือให้พักการทำงานหลักเอาไว้ก่อนเพื่อไปทำงานที่เร่งด่วนหรือสำคัญกว่า โดยการอินเตอร์รัปต์นั้นมีสองลักษณะ คือการอินเตอร์รัปต์จากภายใน เป็นการนับเวลาของ Timer เมื่อ Timer ได้ทำการนับจนถึงค่าสูงสุดที่จะนับได้ หรือค่าที่เราเขียนโปรแกรมสั่งงานไว้ว่าให้นับเพียงเท่านี้ เรียกว่า Overflow และเมื่อเกิดการ Overflow ตัวTimer จะส่งสัญญาณอินเตอร์รัปต์ บอก CPU ว่าได้นับถึงค่า Overflow แล้ว และจะกลับไปเริ่มต้นนับใหม่ ที่ค่าเริ่มต้นซึ่งโดยปกติแล้วเป็น 0 แต่เรายังสามารถเขียนโปรแกรมกำหนดค่าเริ่มต้นในการนับเองก็ได้

การอินเตอร์รัปต์อีกแบบหนึ่งเป็น การอินเตอร์รัปต์จากภายนอก โดยการรับค่าเข้ามาทาง พอร์ตอินพุตและต้องเป็นขาที่ใช้ผู้ผลิตกำหนดมาให้สามารถใช้งานได้เท่านั้น หากเทียบดูจากดาต้าชีทจะเขียนไว้ว่า INT และไอซีแต่ละเบอร์มีขาใช้งานอินเตอร์รัปต์ไม่เท่ากัน บางเบอร์อาจมี 1ขา 2ขา หรือ 3ขาหรืออาจมากกว่าต้องดูจากดาต้าชีทของไอซีเบอร์นั้นๆ การใช้งานสามารถสั่งงานได้ด้วยต่อสวิตซ์กดสั่งงาน หรืออาจจะรับค่ามา

จากเซ็นเซอร์ดิจิทัลต่างๆ ไมโครคอนโทรลเลอร์ที่ใช้ในบอร์ด UNO R3 นั้นมีขาอินเตอร์รัปต์มาให้ 2 ขา คือขา 2 และ ขา 3 เท่านั้น

ในการใช้งานอินเตอร์รัปต์จาก Timer นั้นต้องมีคู่มือการใช้งานจากดาต้าชีท ของไอซีไมโครคอนโทรลเลอร์เบอร์นั้น เพื่อเปรียบเทียบการตั้งค่า เพราะไมโครคอนโทรลเลอร์แต่ละเบอร์นั้น มีรีจิสเตอร์ที่ไม่เหมือนกัน ดาต้าชีทจึงเป็นสิ่งจำเป็นมากในการใช้งานไมโครคอนโทรลเลอร์





มีสิ่งที่ต้องทำความเข้าใจเกี่ยวกับการนับ ระหว่าง เคาน์เตอร์ Counter และ ไทม์เมอร์ Timer

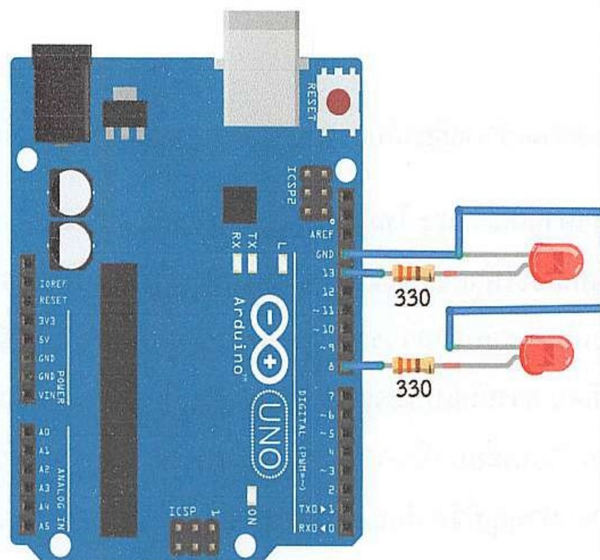
ถึงแม้ว่า เคาน์เตอร์ และ ไทม์เมอร์ นั้นเป็นการนับเหมือนกัน แต่เป็นการนับที่แตกต่างกันต้องทำความเข้าใจและแยกแยะให้ถูกว่า ในงานที่ทำนั้นควรจะใช้การนับแบบเคาน์เตอร์หรือไทม์เมอร์ การนับแบบเคาน์เตอร์นั้นเป็นการนับจำนวนรวมทั้งหมด ยกตัวอย่างเช่น หากต้องการนับว่ามีคนเข้ามาทำบุญที่วัดในวันนี้ทั้งหมดกี่คน การนับแบบนี้จะเรียกว่าเคาน์เตอร์ ส่วนการนับแบบไทม์เมอร์นั้นเป็นการนับโดยมีเวลาเข้ามาเป็นตัวเปรียบเทียบ เป็นการสุ่มนับตามเวลาที่กำหนด ยกตัวอย่างเช่น ต้องการนับว่าในหนึ่งชั่วโมงนั้นมีคนเข้ามาทำบุญที่วัด กี่คน นี่คือการนับแบบ ไทม์เมอร์ Timer เป็นการนับโดยมีเวลามาเป็นตัวกำหนด อย่างเช่น เกจวัดความเร็วของรถยนต์ เป็นการนับว่าความเร็วที่เคลื่อนที่อยู่นั้น ในหนึ่งชั่วโมงจะสามารถเคลื่อนที่ไปได้กี่กิโลเมตร หรือการนับรอบการหมุนของมอเตอร์ว่า ใน 1 นาทีจะหมุนได้จำนวนรอบกี่รอบ เป็นการนับจำนวนรอบที่มอเตอร์หมุนได้ภายในเวลา 1 นาที แบบนี้เป็นการนับแบบไทม์เมอร์

ฟังก์ชันในหมวดหมู่ของการ Interrupt

ฟังก์ชันในหมวดหมู่ของการ Interrupt มีอยู่สองฟังก์ชัน

interrupts();

Interrupts() เป็นฟังก์ชันที่ใช้สำหรับเปิดใช้งานอินเตอร์รัปต์ จากการนับเวลาของไทม์เมอร์ภายใน ใช้กับงานที่ต้องการใช้งานไมโครคอนโทรลเลอร์ตัวเดียวแต่ใช้งานหลากหลายหน้าที่ในเวลาเดียวกัน แต่การใช้งานค่อนข้างยุ่งยากเล็กน้อย เพราะต้องตั้งค่าการใช้งานต่างๆตามดาต้าชีทของไมโครคอนโทรลเลอร์ตัวนั้นๆ ซึ่งดาวน์โหลดได้จากบริษัทผู้ผลิต จากตัวอย่างเป็นการสั่งงานไฟกระพริบ สองดวง ซึ่งดวงหนึ่งกระพริบจากการอินเตอร์รัปต์ แต่ดวงหนึ่งนั้นจะกระพริบตามโปรแกรมหลักในฟังก์ชัน loop() แสดงให้เห็นว่าการสั่งงานให้กระพริบนั้นถูกสั่งมาจากคนละที่ อย่างอิสระ โดยไม่เกี่ยวข้องกัน โดยกำหนดให้ LED ภายในบอร์ดที่ต่ออยู่กับขา 13 นั้นกระพริบเมื่อมีการอินเตอร์รัปต์ แต่ led2 ที่ต่ออยู่กับขา 8 นั้นถูกสั่งงานให้กระพริบทุก 300ms ตามการทำงานใน loop



```
#define led 13                     // กำหนดชื่อ led แทนขา 13
#define led2 8                     // กำหนดชื่อ led2 แทนขา 8
void setup()
{
  pinMode(led, OUTPUT);
  pinMode(led2, OUTPUT);
  noInterrupts();                  // ปิดการใช้งานอินเตอร์รัปต์ทั้งหมด
  TCCR1A = 0;
  TCCR1B = 0;
  TCNT1  = 0;
  OCR1A = 31250;                   // compare match register 16MHz/256/2Hz
  TCCR1B |= (1 << WGM12);          // CTC mode
  TCCR1B |= (1 << CS12);           // กำหนดค่า prescaler 256
  TIMSK1 |= (1 << OCIE1A);         // เปิดอินเตอร์รัปต์จากไทม์เมอร์
  interrupts();                    //เปิดใช้งานอินเตอรัปต์ทั้งหมด
}

ISR(TIMER1_COMPA_vect)             // หากมีการอินเตอร์รัปต์
{
  digitalWrite(led, digitalRead(led) ^ 1);    // กลับสถานะ led ที่ขา 13
}
void loop()
{
  digitalWrite(led2,HIGH);           //ให้ led ที่ขา 8 ติด
  delay(300);                        //หน่วงเวลา 300 มิลลิวินาที
  digitalWrite(led2,LOW);            //ให้ led ที่ขา 8 ดับ
  delay(300);                        //หน่วงเวลา 300 มิลลิวินาที
}
```

ฟังก์ชัน noInterrupts()

noInterrupts();

เป็นฟังก์ชันที่ใช้งานคู่กับ interrupt เพื่อใช้ในการปิดการใช้งานอินเตอร์รัปต์ทั้งหมด

ฟังก์ชันในหมวด การใช้งาน External Interrupt หรือการอินเตอร์รัปต์จากภายนอก มีอยู่ 2 ฟังก์ชัน

ฟังก์ชัน attachinterrupts()

attachinterrupts();

เป็นการใช้งานอินเตอร์รัปต์จากภายนอก เป็นการรับสัญญาณเข้ามาจากขาที่รองรับการอินเตอร์รัปต์ซึ่งในบอร์ด UNO R3 มีมาให้สองขา คือ ตำแหน่งขา 2 และ ขา 3 โดยไมโครคอนโทรลเลอร์จะกระโดดไปทำงานตามคำสั่งอินเตอร์รัปต์ ทันทีที่มีสัญญาณเข้ามาที่ขา 2 หรือ ขา 3 แล้วแต่การเรียกใช้งาน เมื่อเสร็จงานแล้วจึงจะกลับไปทำงานหลักต่อไป

```
#define led 8                        //ประกาศให้ขา8 ขับ LED
#define btn 2                        //ประกาศให้ขา2 เป็นปุ่มกด

void setup() {
  pinMode(led,OUTPUT);               //ประกาศให้ขา8 ทำงานเป็นเอาต์พุต
  pinMode(btn,INPUT_PULLUP);  //ประกาศให้ขา2 ทำงานเป็นอินพุต

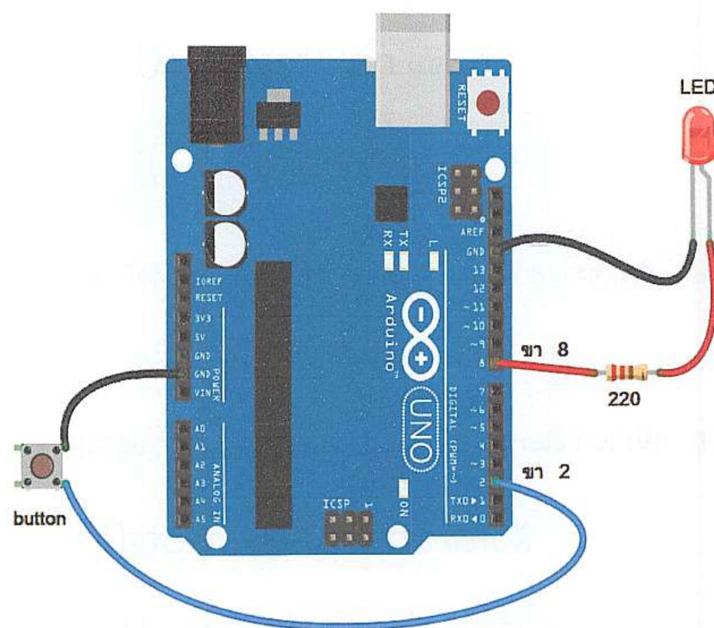
  attachInterrupt(digitalPinToInterrupt(btn),check_SW,LOW);

  //ประกาศเรียกใช้งานฟังชั่นอินเตอร์รัปต์ให้ทำงานเมื่อขา 2 มีสถานะเป็น LOW
}

 void check_SW(){    //ฟังชั่น Interrupt
  digitalWrite(led,!digitalRead(led));
 }

void loop() {

}
```



ตัวอย่างการใช้งานอินเตอร์รัปต์ โดยต่อสวิตซ์ไว้ที่ขา 2 หากมีการกดสวิตซ์ led จะติดสว่างขึ้นมา และหากกดสวิตซ์อีกครั้งหนึ่ง led จะดับลงอย่างเดิม จากตัวอย่างนี้แสดงให้เราเห็นอีกด้วยว่าเราสามารถสั่งงาน เปิดไฟ หรือ ปิดไฟ ได้ด้วยสวิตซ์ปุ่มกดเพียงตัวเดียวเท่านั้น เป็นโปรแกรมทดสอบการอินเตอร์รัปต์ที่ง่ายที่สุด

แต่ว่า จากตัวอย่างโค้ดโปรแกรมที่ผ่านมานั้น เราจะพบปัญหาหนึ่งคือเมื่อมีการกดปุ่มแล้วบางที LED ก็อาจจะติด หรือบางทีก็อาจจะไม่ติด หรือติดๆดับๆ อันเป็นผลมาจากหน้าสัมผัสของสวิตซ์ปุ่มกดที่เป็นแผ่นโลหะบางๆ เมื่อมีการกดปุ่มลงไปด้วยมือเรา มันก็มีการกระเด้งกระดอนแตะกันมากกว่า 1 ครั้ง เพราะไมโครคอนโทรลเลอร์มันทำงานเร็วมาก มันจึงตรวจจับจำนวนครั้งได้ทั้งหมด เรากดครั้งเดียวก็จริง แต่กว่าหน้าสัมผัสมันจะต่อกันสนิท อาจจะตรวจจับได้ หลายครั้ง ดังนั้นในการใช้งานจริงจึงต้องมีการแก้ไขปัญหาดังกล่าว เรียกว่าการ Debounce ดังที่เคยกล่าวมาแล้ว แต่ประเด็นอยู่ที่การ Debounce ด้วยซอฟต์แวร์นั้นเราใช้การหน่วงเวลาไว้ด้วย delay() หากเป็นการใช้งานใน loop ธรรมดาย่อมได้ผล แต่สำหรับในฟังก์ชันInterrupt การหน่วงเวลาด้วยฟังก์ชัน delay() นั้นจะไม่มีผล เพราะในฟังก์ชัน Interrupt นั้นจะไม่สามารถใช้งาน delay() ได้ ถึงแม้ในขั้นตอนคอมไพล์โปรแกรมจะไม่มีเออเร่อ แต่ตอนทำงานจริงมันจะไม่มีการหน่วงเวลาให้ แนวทางแก้ไขคือ ให้ใช้การนับเวลาด้วยวิธีอื่นเช่นการใช้งานนับเวลาจากไทม์เมอร์ซึ่งอาจจะยากไปสำหรับมือใหม่วิธีที่ง่ายคือใช้ประโยชน์จากฟังก์ชันสำเร็จรูป เช่น millis(); ดังตัวอย่างต่อไปนี้

```
#define led 8
#define btn 2
int debounce = 200;
void setup() {
  pinMode(led,OUTPUT);
  pinMode(btn,INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(btn),check_SW,LOW);
}
 void check_SW(){
  static unsigned long t = 0;
  unsigned long lt = millis();
  if(lt - t > debounce){
  digitalWrite(led,!digitalRead(led));
   }
   t = lt;
 }
void loop(){}
```

ฟังก์ชัน detachinterrupts()

detachinterrupts();

detachInterrupts() เป็นฟังก์ชันที่มีไว้ยกเลิกการใช้งานของ attachInterrupt()

ฟังก์ชันในหมวดหมู่ของการติดต่อสื่อสาร Communication

ฟังก์ชันในหมวดหมู่ของ Communication เป็นฟังก์ชันสำเร็จรูปของการสื่อสารคือฟังก์ชัน Serial()

ฟังก์ชัน Serial

Serial

Serial เป็นฟังก์ชันที่ใช้สำหรับการติดต่อสื่อสารเพื่อรับ-ส่งข้อมูลแบบอนุกรม กับอุปกรณ์อย่างอื่น เช่น คอมพิวเตอร์ ,Bluetooth หรือ WIFI เป็นต้น ฟังก์ชัน Serial นี้ยังมีฟังก์ชันย่อยอีกเยอะมาก แต่ที่เราใช้งานกันแบบธรรมดาทั่วไปในเบื้องต้นนั้นมีอยู่ไม่กี่ฟังก์ชัน ในการรับหรือส่งข้อมูลอนุกรมนั้นสิ่งที่ต้องคำนึงถึงเป็นอันดับแรกก็คือความเร็วในการรับ-ส่งข้อมูล เพราะการเชื่อมต่ออุปกรณ์ที่จะให้รับส่งข้อมูลระหว่างกันนั้นจะต้องตั้งค่าความเร็วในการรับส่งให้เท่ากัน จึงจะสามารถรับ-ส่งข้อมูลระหว่างกันได้

Serial.begin(); เป็นการตั้งค่าความเร็วในการรับ-ส่งข้อมูล โดยใส่ตัวเลขค่าความเร็วลงในวงเล็บ มีหน่วยเป็น บิต ต่อวินาที เช่น **Serial.begin(9600);** ดังนั้นก่อนที่จะเรียกใช้งานการรับส่งข้อมูล สิ่งที่จะต้องทำอันดับแรกคือ การตั้งค่าความเร็วในการรับส่งข้อมูลด้วยฟังก์ชัน **Serial.begin()**

ในการเชื่อมต่อใช้งานจริงควรดูว่าอุปกรณ์ที่จะเชื่อมต่อด้วยนั้นตั้งค่าความเร็วไว้ที่เท่าไหร่ควรตั้งให้ตรงกันจึงจะสามารถสื่อสารกันได้โดยค่าความเร็วที่ใช้ทั่วไปจะมีตั้งแต่

300,1200,2400,4800,9600,12200,38400,57600,74880,115200 เป็นต้น

Serial.print() เป็นการส่งข้อมูลแบบเป็นตัวอักขระหรือข้อความ

Serial.println() เป็นการส่งข้อมูลแบบเป็นตัวอักขระหรือข้อความ แต่จะมีการขึ้นบรรทัดใหม่ให้ด้วย

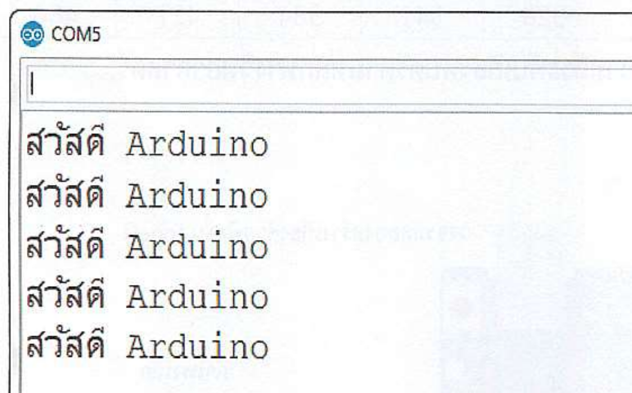
Serial.write() เป็นการส่งข้อมูลชนิดตัวเลข โดยการส่งจะทำการส่งเป็นไบต์ หรือส่งต่อเนื่องเป็นชุด

Serial.read() เป็นการอ่านข้อมูลที่รับเข้ามาได้

Serial.readByte() เป็นการรับข้อมูลอักขระ การทำงานจะสิ้นสุดก็ต่อเมื่อรับข้อมูลได้จนครบ

ในการส่งข้อความไปที่คอมพิวเตอร์นั้น ไม่ต้องมีการต่อวงจรอะไรเพิ่มเติม เพราะว่าบอร์ด Arduino นั้นได้ออกแบบมาให้สามารถเชื่อมต่อกับคอมพิวเตอร์ได้โดยผ่านพอร์ต USB ของคอมพิวเตอร์ได้อยู่แล้ว ดังนั้นเราจึงสามารถเขียนโปรแกรมส่งข้อความออกไปได้ทันทีด้วยฟังก์ชัน Serial.println() ตัวอย่างโปรแกรมการส่งข้อความไปที่คอมพิวเตอร์

```
void setup() {
  Serial.begin(9600);                   //กำหนดค่าความเร็ว 9600
}
void loop() {
   Serial.println("สวัสดี Arduino");     //ส่งข้อความ "สวัสดี Arduino"
   delay(1000);                         //หน่วงเวลาไว้ 1 วินาที
}
```



ภาพแสดงข้อความที่รับได้

ฟังก์ชัน tone()

tone()

tone เป็นฟังก์ชันที่มีไว้สำหรับสร้างเสียงต่างๆโดยจะส่งเป็นสัญญาณสแควร์เวฟ ออกมาตามความถี่ ที่กำหนด ด้วยค่าดิวตี้ไซเคิล 50% ในการสร้างเสียงด้วยฟังก์ชันนี้ จะต้องมีการกำหนดระยะเวลา ในการเกิดเสียงด้วย เพราะไม่เช่นนั้นจะส่งสัญญาณเสียงตลอดเวลา หรืออาจจะสั่งหยุดเสียงได้ด้วยฟังก์ชัน noTone() การใช้งานฟังก์ชันเสียงนี้สามารถที่จะต่อลำโพงเล็กๆเข้ากับขาของบอร์ดได้โดยตรง แต่จะส่งสัญญาณเสียงออกได้เพียงทีละขาเท่านั้น ไม่สามารถส่งสัญญาณออกพร้อมกันครั้งละหลายๆขาได้

การใช้งานนั้นง่ายมาก เพียงแค่ใส่ **(ชื่อขาที่จะใช้,ความถี่เสียงที่ต้องการ,ระยะเวลา)** ลงในวงเล็บเท่านั้น

ชื่อขาคือชื่อขาที่พิมพ์อยู่บนบอร์ด UNO R3

ความถี่เสียงใส่เป็นตัวเลข มีหน่วยเป็น Hz มีค่าตั้งแต่ 31Hz ถึง 65,535 Hz

ระยะเวลามีหน่วยเป็นมิลลิวินาที

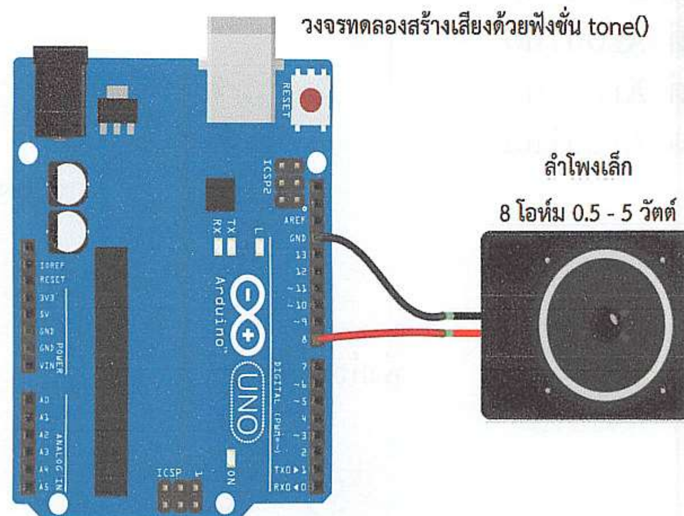
```
tone(ขา,ความถี่เสียง,ระยะเวลา)
```

```
tone(8,256,500)
```

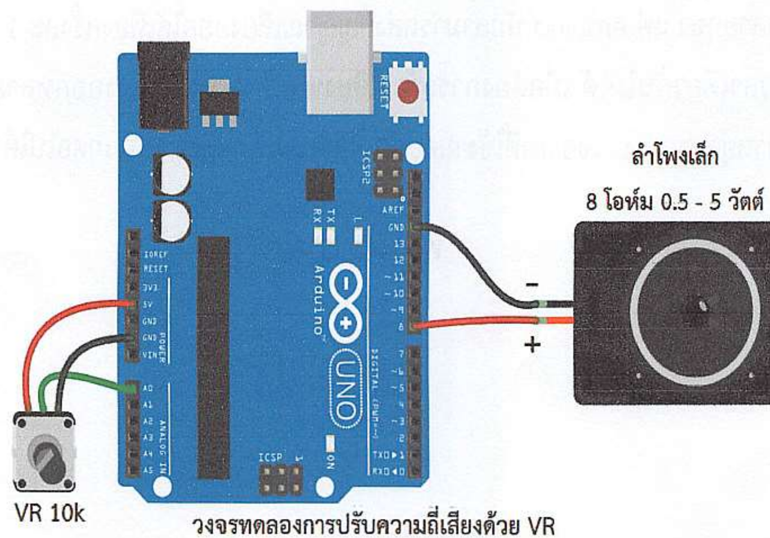
ตัวอย่างการส่งความถี่เสียงออกขา 8 ด้วยความถี่ 256Hz เป็นเวลา 500ms (ครึ่งวินาที)

เสียงดนตรี	โด	เร	มี	ฟา	ซอล	ลา	ที	โด
ความถี่ (Hz)	256	288	320	341	384	427	480	512

ตารางความถี่ระดับเสียงดนตรีตามหลักทางวิทยาศาสตร์



```
unsigned int sound[]={256,288,320,341,384,427,480,512};
void setup() {
  pinMode(8,OUTPUT);
}
void loop() {
  for(int i=0;i<8;i++){
    tone(8,sound[i],500);
    delay(500);
  }
  delay(1000);
}
```



วงจรทดลองการปรับความถี่เสียงด้วย VR

//โปรแกรมตัวอย่าง ปรับความถี่เสียงด้วย VR

```
void setup() {
     pinMode(8,OUTPUT);          //กำหนดให้ขา 8 ทำงานเป็นเอาต์พุต
}

void loop() {
 int vr = analogRead(A0);        //อ่านค่าการปรับ VR จากขา A0
 vr = map(vr,0,1023,256,2000); //แปลงค่าที่อ่านได้เป็นความถี่เสียง 256Hz-2KHz
 tone(8,vr,500);       //ส่งสัญญาณเสียงออกขา 8 ด้วยความถี่ตามการปรับ VR
 delay(800);           //หน่วงเวลา 800 มิลลิวินาที
}
```

ฟังก์ชัน notone()

notone()

เป็นฟังก์ชันที่มีไว้สำหรับหยุดเสียงของฟังก์ชัน tone() การใช้งานฟังก์ชันนี้เพียงแค่ใส่ชื่อขา ที่ต้องการให้หยุดเล่นเสียงลงในวงเล็บ เช่นหากต้องการให้หยุดเล่นเสียงที่ขา 8 ก็เขียนว่า noTone(8);

noTone(ชื่อขา) ตัวอย่างใช้งานจริง → noTone(8);

ฟังก์ชันนี้จะมีประโยชน์ในการหยุดเล่นเสียงที่กำลังเล่นอยู่ และโปรแกรมที่ต้องการสร้างความถี่เสียงส่งออกหลายๆขา แต่ Arduino นั้นสามารถส่งสัญญาณเสียงออกได้เพียงครั้งละ 1 ขา เท่านั้นจะส่งพร้อมกันทุกขาในเวลาเดียวกันไม่ได้ เมื่อต้องการสร้างเสียงหลายๆความถี่ให้ส่งออกหลายขา จึงต้องใช้ฟังก์ชัน noTone() มาหยุดการทำงานของขาที่ใช้งานอยู่ก่อน จึงจะสั่งเล่นเสียง ขาอื่นๆต่อไปได้

ฟังก์ชัน map()

map ()

ฟังก์ชัน map() เป็นฟังก์ชันสำเร็จรูปของ Arduino ที่มีไว้สำหรับการแปลงค่าตัวเลข จากช่วงตัวเลขหนึ่ง ไปเป็นตัวเลขอีกช่วงหนึ่ง เช่น ต้องการจะแปลงค่า 0-99 ให้เหลือเป็นเพียงแค่ช่วงสั้น 0-9

ฟังก์ชัน map() จะทำการแปลงค่าช่วงของตัวเลขให้โดยอัตโนมัติเพียงแค่ใส่ค่าเริ่มต้น และค่าที่ต้องการจะให้เป็น ลงในวงเล็บ

ก่อนแปลง		หลังแปลง
0	→	0
0 - 9	→	1
10 - 19	→	2
20 - 29	→	3
30 - 39	→	4
40 - 49	→	5
50 - 59	→	6
60 - 69	→	7
70 - 79	→	8
80 - 99	→	9

map (ตัวแปร,ค่าเริ่มต้น,ค่าสูงสุด,ค่าเริ่มต้นที่ต้องการ,ค่าสูงสุดที่ต้องการ)

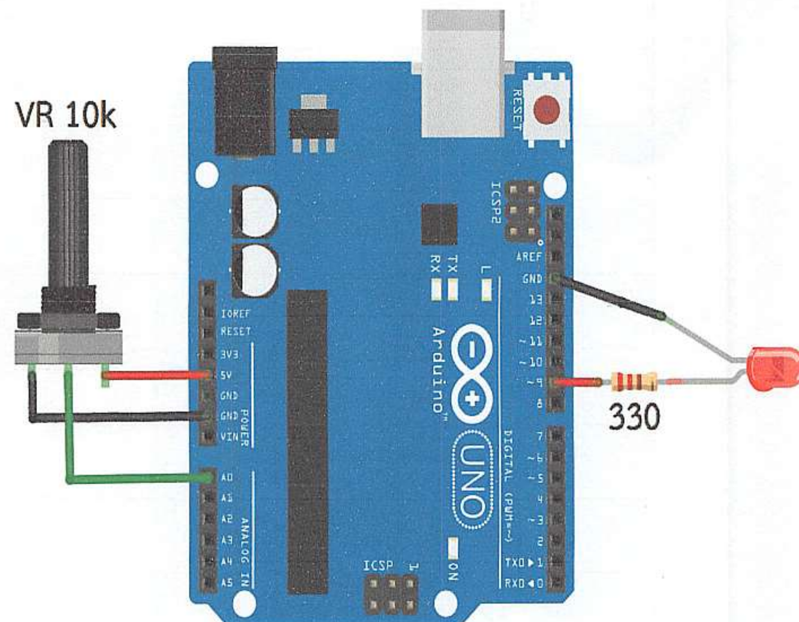
x = map (x,0,99,0,9)

ตัวอย่างการแปลงค่าตัวเลข 0-99 ให้เป็น 0-9

โปรแกรมอ่านค่าจากการปรับ VR

```
#define led 9                       //กำหนดชื่อ led ขึ้นมาแทนขา 9
#define vr A0                       //กำหนดชื่อ vr ขึ้นมาแทนขา A0
unsigned int v;                     //ประกาศตัวแปรเพื่อใช้เก็บค่าจากการปรับ vr
unsigned int t;                     //ประกาศตัวแปรเพื่อใช้เป็นค่าหน่วงเวลา
void setup() {
  pinMode(led,OUTPUT);              //กำหนดให้ขา 9 เป็นเอาต์พุตเพื่อขับ LED
}
void loop() {
  v = analogRead(vr);               //อ่านค่าจากการปรับ vr ที่ขา A0 ในเก็บในตัวแปร v
  t = map(v,0,1023,50,2000);        //แปลงค่าที่ตัวเลขที่อ่านจากตัวแปร v 0-1023 ด้วยฟังชั่น map()
                                    //ให้เป็นช่วงตัวเลข 50-2000 เพื่อนำไปกำหนดค่าเวลาเป็นมิลลิวินาที
  digitalWrite(led,HIGH);           //ให้หลอด LED ติด
  delay(t);                         //หน่วงเวลาตามค่าที่รับมาจาก vr
  digitalWrite(led,LOW);            //ให้หลอด LED ดับ
  delay(t);                         //หน่วงเวลาตามค่าที่รับมาจาก vr
}
```

โปรแกรมอ่านค่าจากการปรับ VR มาแปลงเป็นค่าเวลาเพื่อใช้กำหนดการกระพริบของหลอด LED

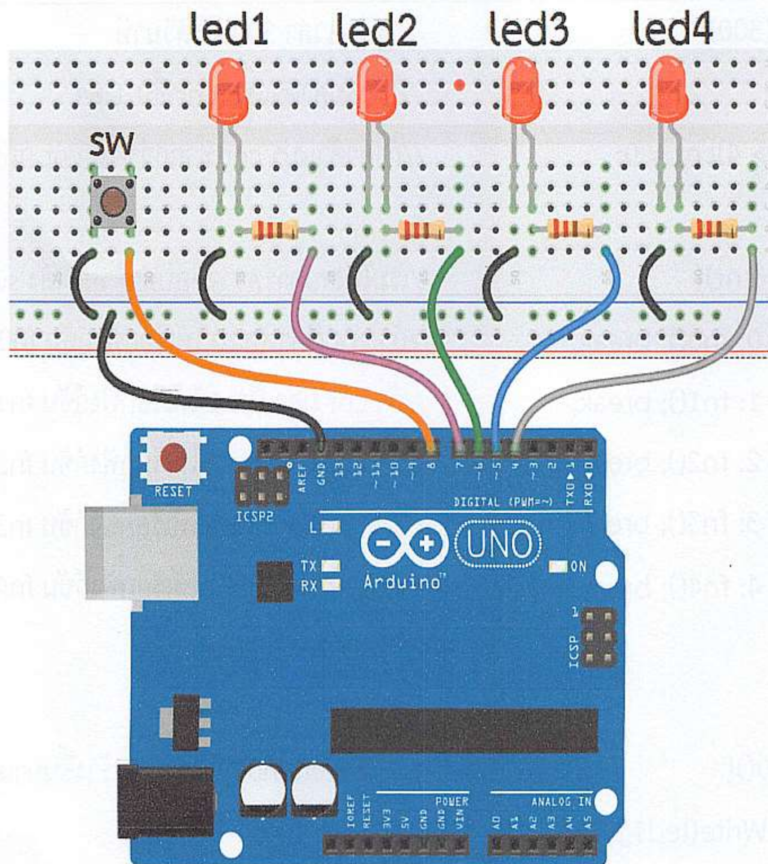


วงจรไฟกระพริบปรับความเร็วด้วย VR ใช้ฟังก์ชัน analogRead() และฟังก์ชัน map()

สิ่งที่ผู้อ่านจะได้ประโยชน์จากโครงงานไฟกระพริบปรับความเร็วในการกระพริบด้วยการปรับ VR นี้ จะแสดงให้เห็นถึงวิธีการอ่านค่าสัญญาณจากพอร์ตอินพุตด้วยฟังก์ชัน **analogRead()** และการแปลงข้อมูลด้วยการแบ่งค่าช่วงตัวเลขที่อ่านได้จาก **analogRead()** ด้วยฟังก์ชัน **map()** เพื่อแปลงค่าตัวเลขที่อ่านได้จากการปรับ VR เอามาแปลงเป็นค่าเวลาในการกระพริบของหลอด LED

การสร้างฟังก์ชัน และการเรียกใช้ฟังก์ชันที่เราสร้างเอง

จากที่ได้กล่าวไว้แต่ตอนแรกแล้วว่านอกจากฟังก์ชันหลักที่ Arduino มีมาให้ คือ ฟังก์ชันที่ชื่อว่า setup() และฟังก์ชัน loop() แต่เราก็ยังสามารถที่จะเขียนฟังก์ชันขึ้นมาใช้งานเองได้ด้วย จากคำถามต่างๆของหลายๆคน ที่เริ่มศึกษาการเขียนโปรแกรมด้วย Arduino ผู้ที่เพิ่งจะเริ่มศึกษาการเขียนโปรแกรมนั้นจะมีปัญหากันมากเช่น จะทำอย่างไรให้กดสวิตซ์ 3 ครั้ง หรือ 5 ครั้ง แล้วค่อยสั่งเปิดไฟ หรืออยากจะให้โปรแกรมที่เขียนขึ้นมานั้นสามารถเลือกโหมดการทำงานได้ หรือบางคนอาจจะรู้วิธีการเขียนฟังก์ชันขึ้นมาใช้เองแล้ว แต่ไม่รู้จะเรียกใช้งานได้อย่างไร ปัญหาต่างๆเหล่านี้จะหมดไปเมื่อได้เรียนรู้และทำความเข้าใจถึงโครงสร้างการทำงานของภาษา C อย่างแท้จริง ด้วยตัวอย่างง่ายๆต่อไปนี้



//โปรแกรมตัวอย่างการใช้งานสวิตซ์เลือกฟังก์ชัน

```
#define led1 7                    //led1 หมายถึง ขา 7
#define led2 6                    //led2 หมายถึง ขา 6
#define led3 5                    //led3 หมายถึง ขา 5
#define led4 4                    //led4 หมายถึง ขา 4
#define sw 8                      //sw หมายถึง ขา 8
unsigned int cnt = 0;             //สร้างตัวแปร cnt เพื่อใช้ในการนับการกดสวิตซ์
void setup() {                    //เริ่มฟังก์ชัน setup()
  pinMode(led1,OUTPUT);           //ให้ขา 7 ทำงานเป็นเอาต์พุต
  pinMode(led2,OUTPUT);           //ให้ขา 6 ทำงานเป็นเอาต์พุต
  pinMode(led3,OUTPUT);           //ให้ขา 5 ทำงานเป็นเอาต์พุต
  pinMode(led4,OUTPUT);           //ให้ขา 4 ทำงานเป็นเอาต์พุต
  pinMode(sw,INPUT_PULLUP);       //ให้ขา 8 ทำงานเป็นอินพุตและเปิดใช้งาน R PULLUP
}                                 //จบฟังก์ชัน setup()
void loop() {                     //เริ่มฟังก์ชัน loop()
if(digitalRead(sw) == 0){         //หากมีการกดสวิตซ์ ทำให้ขา 8 มีสถานะเป็น 0
```

```
  delay(300);                    //หน่วงเวลา 300 มิลลิวินาที
  cnt++;                         //ให้เพิ่มค่าตัวแปร cnt ขึ้น 1 ค่า
  if(cnt > 4)cnt = 0;            //ถ้าค่าตัวแปร cnt มากกว่า ให้ cnt กลับไปมีค่าเป็น 0
}
  switch(cnt){                   //เริ่มใช้งานการตรวจสอบค่าด้วยคำสั่ง switch...case
    case 0: fn0(); break;        //ถ้า cnt มีค่าเป็น 0 ให้เรียกฟังก์ชัน fn0() ขึ้นมาทำงาน
    case 1: fn1(); break;        //ถ้า cnt มีค่าเป็น 1 ให้เรียกฟังก์ชัน fn1() ขึ้นมาทำงาน
    case 2: fn2(); break;        //ถ้า cnt มีค่าเป็น 2 ให้เรียกฟังก์ชัน fn2() ขึ้นมาทำงาน
    case 3: fn3(); break;        //ถ้า cnt มีค่าเป็น 3 ให้เรียกฟังก์ชัน fn3() ขึ้นมาทำงาน
    case 4: fn4(); break;        //ถ้า cnt มีค่าเป็น 4 ให้เรียกฟังก์ชัน fn4() ขึ้นมาทำงาน
  }                              //สิ้นสุดคำสั่ง switch...case
}                                //จบฟังก์ชัน loop()
void fn0(){                      //เริ่มฟังก์ชัน fn0 ******** สร้างขึ้นมาเอง *********
  digitalWrite(led1,LOW);        //สั่งให้ led1 ดับ
  digitalWrite(led2,LOW);        //สั่งให้ led2 ดับ
  digitalWrite(led3,LOW);        //สั่งให้ led3 ดับ
  digitalWrite(led4,LOW);        //สั่งให้ led4 ดับ
}                                //สิ้นสุดฟังก์ชัน fn0
void fn1(){                      //ฟังก์ชัน fn1 ******** สร้างขึ้นมาเอง *********
  digitalWrite(led1,HIGH);       //สั่งให้ led1 ติด
  digitalWrite(led2,LOW);        //สั่งให้ led2 ดับ
  digitalWrite(led3,LOW);        //สั่งให้ led3 ดับ
  digitalWrite(led4,LOW);        //สั่งให้ led4 ดับ
}                                //สิ้นสุดฟังก์ชัน fn1
void fn2(){                      //เริ่มฟังก์ชัน fn2 ******** สร้างขึ้นมาเอง *********
  digitalWrite(led1,LOW);        //สั่งให้ led1 ดับ
  digitalWrite(led2,HIGH);       //สั่งให้ led2 ติด
  digitalWrite(led3,LOW);        //สั่งให้ led3 ดับ
  digitalWrite(led4,LOW);        //สั่งให้ led4 ดับ
}                                //สิ้นสุดฟังก์ชัน fn2
void fn3(){                      //เริ่มฟังก์ชัน fn3
```

```
  digitalWrite(led1,LOW);          //สั่งให้ led1 ดับ
  digitalWrite(led2,LOW);          //สั่งให้ led2 ดับ
  digitalWrite(led3,HIGH);         //สั่งให้ led3 ติด
  digitalWrite(led4,LOW);          //สั่งให้ led4 ดับ
}                                  //สิ้นสุดฟังก์ชัน fn3
void fn4(){                        //เริ่มฟังก์ชัน fn4 ******** สร้างขึ้นมาเอง *********
  digitalWrite(led1,LOW);          //สั่งให้ led1 ดับ
  digitalWrite(led2,LOW);          //สั่งให้ led2 ดับ
  digitalWrite(led3,LOW);          //สั่งให้ led3 ดับ
  digitalWrite(led4,HIGH);         //สั่งให้ led4 ติด
}                                  //สิ้นสุดฟังก์ชัน fn4 **จบโปรแกรม
```

จากตัวอย่างนี้การทำงานคือเมื่อเรากดสวิตซ์โปรแกรมจะทำการเพิ่มค่าตัวแปร cnt ขึ้นครั้งละ 1 ค่า หมายความว่าในตอนแรก cnt จะมีค่าเป็น 0 และโปรแกรมจะวนทำงานอยู่ที่ loop แต่ใน loop นั้นเราได้เขียนคำสั่ง ให้ตรวจสอบการกดสวิตซ์ หากมีการกดสวิตซ์ จะมีการนับค่าตัวแปร cnt เพิ่มขึ้นไปจน cnt มีค่ามากกว่า 4 ก็จะสั่งงานให้ค่าตัวแปร cnt กลับไปมีค่าเริ่มต้นที่ 0 ใหม่ และหลังจากนั้นก็เขียนคำสั่งไว้คอยตรวจสอบค่าของ cnt ด้วยคำสั่ง switch...case หาก cnt มีค่าเป็น 0 ก็จะสั่งงานให้เรียกใช้ฟังก์ชัน fn0 และในฟังก์ชัน fn0 นั้นเขียนสั่งงานไว้ให้ led ดับหมดทุกดวง

เมื่อ cnt มีค่าเท่ากับ 1 ก็จะเข้าทำงานที่ฟังก์ชัน fn1 led1 จะติดเพียงดวงเดียว

เมื่อ cnt มีค่าเท่ากับ 2 ก็จะเข้าทำงานที่ฟังก์ชัน fn2 led2 จะติดเพียงดวงเดียว

เมื่อ cnt มีค่าเท่ากับ 3 ก็จะเข้าทำงานที่ฟังก์ชัน fn3 led3 จะติดเพียงดวงเดียว

เมื่อ cnt มีค่าเท่ากับ 4 ก็จะเข้าทำงานที่ฟังก์ชัน fn4 led4 จะติดเพียงดวงเดียว

และเมื่อกดสวิตซ์อีก 1 ครั้งจะทำให้ตัวแปร cnt มีค่ามากกว่า 4 ซึ่งเราเขียนคำสั่งเอาไว้ว่าหาก cnt มีค่ามากกว่า 4 เมื่อใด ก็ให้ cnt นั้นกลับไปที่ค่าเป็น 0 การทำงานของโปรแกรมนี้ จึงเป็นการประยุกต์ใช้งานสวิตซ์เพื่อนำไปนับค่าตัวแปร แล้วใช้วิธีการตรวจสอบค่าตัวแปร เอาผลของค่าตัวแปรที่ได้ไปใช้สั่งงานเพื่อเลือกโหมดการทำงานตามฟังก์ชันที่เราเขียนขึ้นมาเองได้ ด้วยคำสั่ง **switch...case**

สรุปได้ว่าการเรียกใช้ฟังก์ชันที่เราสร้างขึ้นมาเองนั้นสามารถเรียกใช้ได้จากฟังก์ชัน loop เพราะโปรแกรมจะวนทำงานอยู่ใน loop เมื่อเราเรียกใช้งานฟังก์ชันโปรแกรมก็จะกระโดดออกไปทำงานในฟังก์ชันนั้น เมื่อจบการทำงานก็จะกลับมาทำงานใน loop ต่อไป เว้นแต่ในกรณีที่ฟังก์ชันที่เราสร้างขึ้นมานั้นมี loop ที่โปรแกรมไม่สามารถหลุดออกมาใช้ เช่น while(1) โปรแกรมจะติดอยู่ในนั้นและจะกลับไปทำงานที่ฟังก์ชันหลักไม่ได้ จึงควรต้องระวังในการใช้งานคำสั่ง while และ for ในฟังก์ชันที่เราเขียนขึ้นมาเอง

ส่วนการเขียนฟังก์ชันขึ้นมาเองนั้นเราจะเขียนไว้ก่อนฟังก์ชัน loop() หรือหลังฟังก์ชัน loop() ก็ได้

การแสดงผลด้วยตัวเลข LED 7Segment

ในงานไมโครคอนโทรลเลอร์ทั่วไปนั้น การใช้งานจริงมิใช่แค่เขียนโปรแกรมให้ทำงาน แค่ไฟกระพริบสองดวงหรือไฟวิ่ง การเขียนโปรแกรมไฟกระพริบ หรือ ไฟวิ่ง หรือ คำสั่งในการใช้งานสวิตซ์เพื่อเปิด-ปิด หลอด LED นั้นถูกนำมาใช้เพื่อการทดสอบการเขียนโปรแกรมสั่งงานอินพุต และ เอาต์พุต ในระดับพื้นฐานในการเริ่มต้นเขียนโปรแกรมเท่านั้น ในการนำไปใช้งานจริงนั้นแล้วแต่ผู้ใช้งานว่าจะนำไมโครคอนโทรลเลอร์มาใช้งานเพื่อการใด เป็นเรื่องของบุคคลที่คิดจะสร้างสิ่งต่างๆขึ้นมาตามความคิดและความต้องการของแต่ละคน การใช้งานไมโครคอนโทรลเลอร์ส่วนใหญ่แล้ว สิ่งที่มักหลีกหนีไม่พ้น คือการเขียนโปรแกรมให้เครื่องมือนั้นสามารถติดต่อกับผู้ใช้งานได้อย่างสะดวก ซึ่งนอกจากจะมีสวิตซ์ หรือ เซนเซอร์ต่างๆ เพื่อรับค่าคำสั่งไปสั่งงานแล้ว สิ่งที่จำเป็นอีกอย่างก็คือจอแสดงผล เพราะหากไม่มีจอแสดงผล ผู้ใช้ก็ไม่สามารถรู้ได้เลยว่าเครื่องที่เราสร้างขึ้นมานั้นมีสถานะ การทำงานเป็นอย่างไร ยังทำงานปกติดีอยู่ไหม หรือว่ามันพังไปตั้งนานแล้ว ฮา...

จอแสดงผลที่มักพบทั่วไปในงานไมโครคอนโทรลเลอร์ส่วนใหญ่คือ LED 7Segment เนื่องจากมีราคาไม่แพงมากนัก เราจึงพบเห็นกันได้ทั่วไป และอาจจะมีอยู่แล้วทุกบ้านก็เป็นได้ อย่างเช่นตัวเลขแสดงเลขช่องทีวีที่อยู่หน้าจอเครื่องรับสัญญาณดาวเทียม นาฬิกาดิจิทัล จอแสดงผลเครื่องซักผ้าแบบอัตโนมัติ สิ่งเหล่านี้มีกันอยู่ทุกบ้าน จึงกล่าวได้ว่า ระบบไมโครคอนโทรลเลอร์ทุกวันนี้ มันไม่ใช่เรื่องไกลตัวอีกต่อไป แต่ LED 7Segment นั้นแสดงผลได้เพียงตัวเลข หรืออาจดัดแปลงให้แสดงผล ตัวอักษรภาษาอังกฤษ ได้เพียงไม่กี่ตัวเท่านั้น

โครงสร้างของ LED 7Segment นั้นเป็นหลอด LED 7 หลอด ถูกวางตำแหน่งไว้ให้เป็นเลข 8 โดยกำหนดชื่อเรียกแต่เซ็กเมนต์ หรือ แต่ละส่วนด้วยตัวอักษรภาษาอังกฤษ a b c d e f g และ LED ทั้งหมดมีจุดต่อร่วมกันทุกตัวเรียกว่า คอมม่อน มีอยู่ด้วยกันสองชนิด คือคอมม่อน Anode กับคอมม่อน Cathode

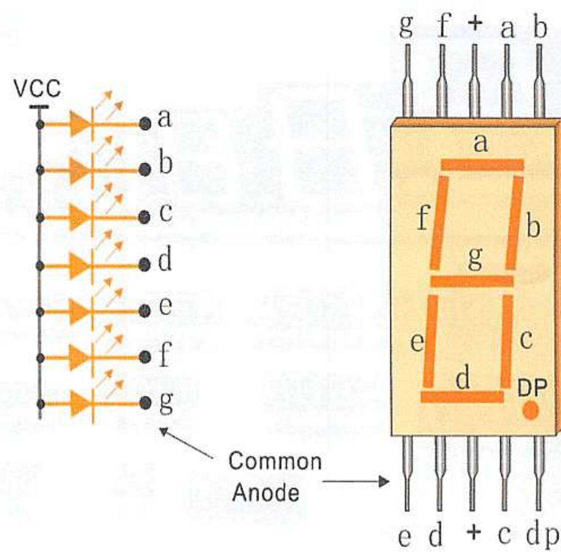
คอมม่อน Anode คือการนำขา Anode ของหลอด LED มาต่อรวมกันทั้งหมด

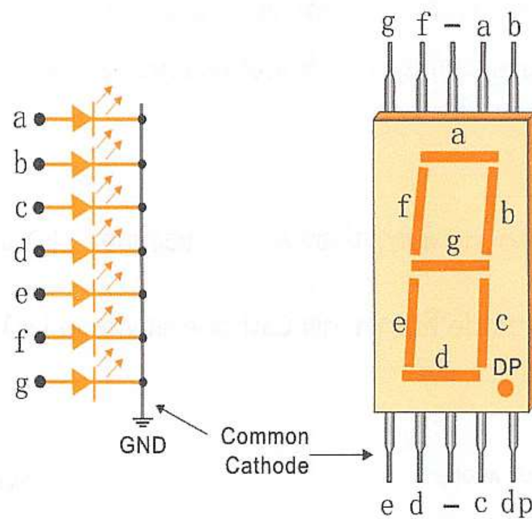
คอมม่อน Cathode คือการนำขา Cathode ของหลอด LED มาต่อรวมกันทั้งหมด

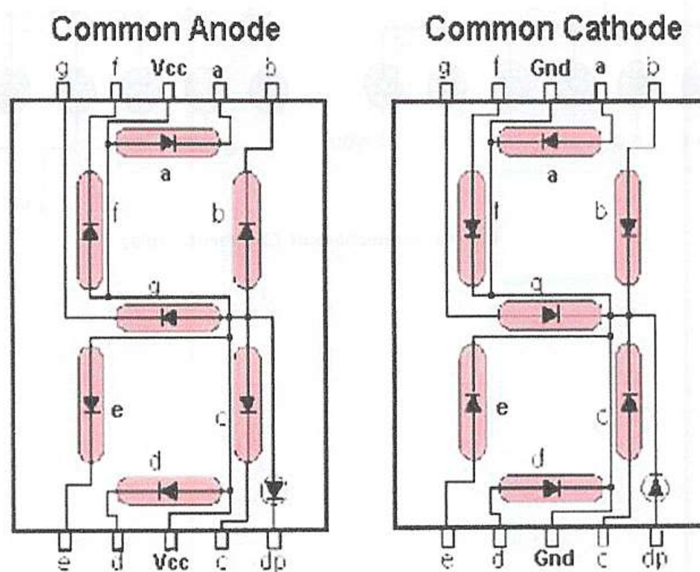
COMMON ANODE

COMMON CATHODE

Internal connections of 7 Segment Display

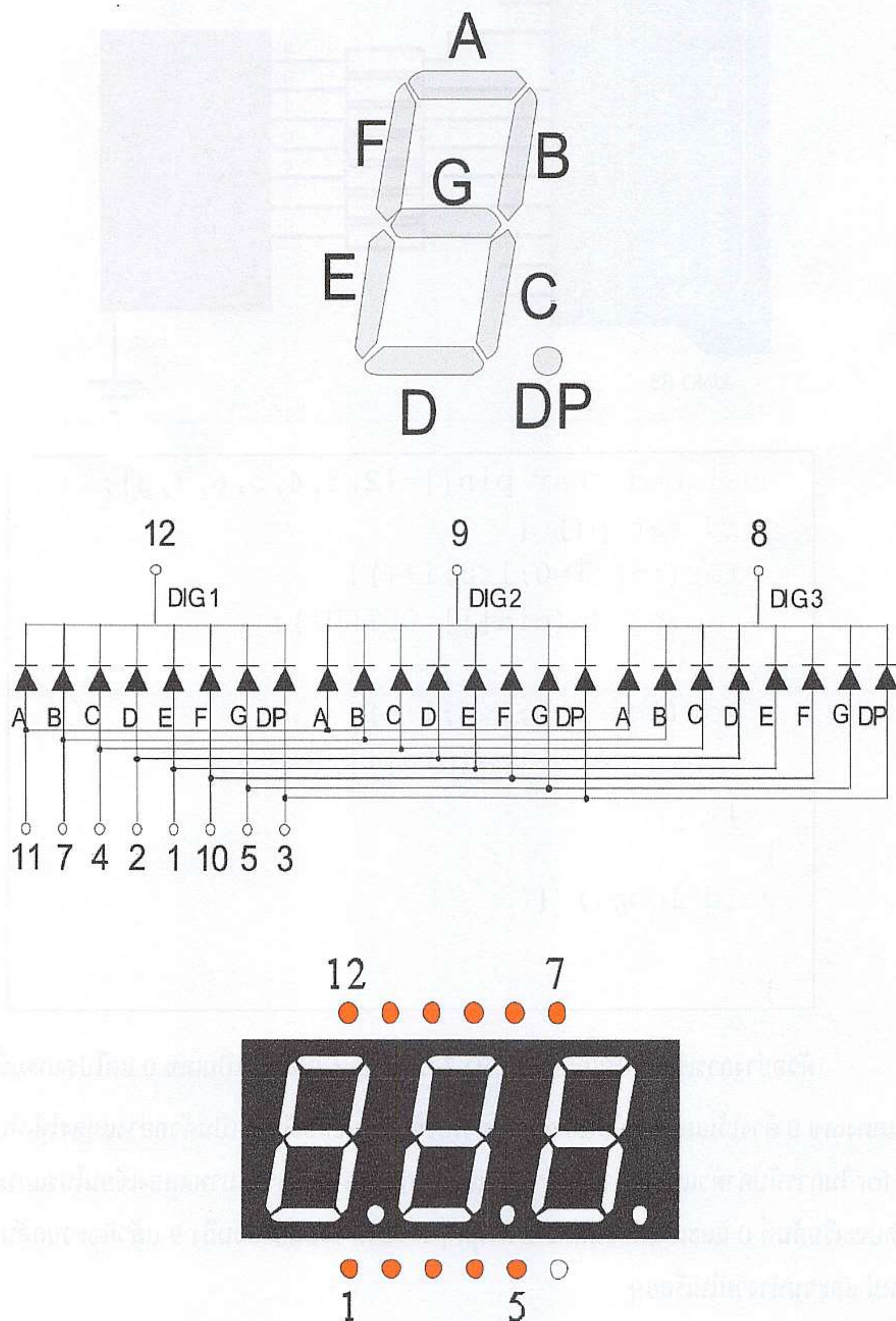






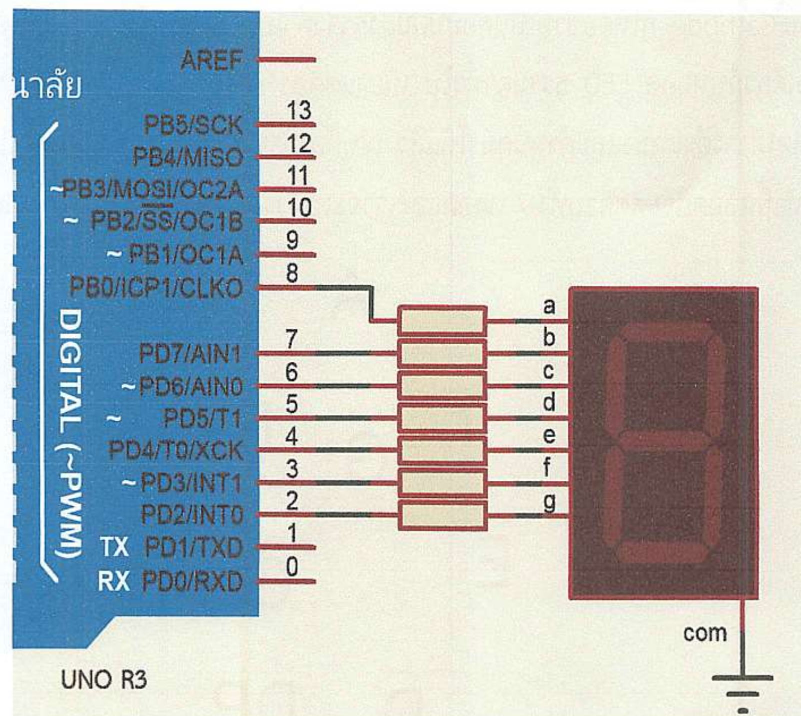
รูปโครงสร้างการวางหลอด LED 7Segment แบบหลักเดียว มาตรฐานทั่วไป

หลักการแสดงผลของ LED 7Segment นั้นไม่ได้มีอะไรซับซ้อน ยกตัวอย่างเช่น LED 7Segment ชนิด คอมม่อน Cathode ก็ให้ต่อขาที่เป็นคอมม่อนลงกราวด์ และเมื่ออยากให้หลอดใดติด ก็ให้เอาไฟไปจ่ายให้หลอดนั้นเช่นเดียวกับหลอด LED ธรรมดาทั่วไป เพียงแต่ต้อง LED 7Segment นั้นเราต้องการแสดงผลเป็นตัวเลขซึ่งผู้ผลิตได้วางตำแหน่งและกำหนดมาให้แล้ว เช่นต้องการให้ติดเป็นเลข 8 เอาไฟไปจ่ายเข้าขา Anode ทุกขา เพียงเท่านี้หลอดก็จะติดครบทั้ง 7 หลอดและเราจะเห็นว่าหลอดที่ติดนั้นแสดงเป็นเลข 8



รายละเอียดและตำแหน่งขาของ led 7segment F5361AH

ตัวอย่างการเขียนโปรแกรมขับ LED 7 Segment แบบหลักเดียว อย่างง่าย




```
unsigned char pin[]={2,3,4,5,6,7,8};
void setup() {
  for(int i=0;i<8;i++){
    pinMode(pin[i],OUTPUT);
  }
  for(int i=0;i<6;i++){
    digitalWrite(pin[i],HIGH);
  }
}
void loop() {

}
```

ตัวอย่างการเขียนโปรแกรมขับ LED 7 Segment ให้แสดงเป็นเลข 0 แต่โปรแกรมนี้ทำได้เพียงแค่แสดงเลข 0 ค้างไว้เฉยๆเท่านั้นไม่สามารถนำไปใช้ประโยชน์อะไรได้ เป็นตัวอย่างแสดงให้เห็นการใช้งานลูป for ในการนับค่าตัวแปรในอาเรย์เท่านั้น และ จากวงจรเดิมนี้เราจะมาทดลองเขียนโปรแกรมนับเลข 0-9 โดยจะเริ่มต้นที่ 0 และเพิ่มค่าขึ้นทีละ 1 ค่าทุกๆ 1 วินาที เมื่อนับไปจนถึง 9 แล้วก็จะวนกลับมาเริ่มต้นที่ 0 ใหม่ และวนทำงานไปเรื่อยๆ

```
//โค้ดโปรแกรมนับเลข 0-9 อย่างง่าย แบบไม่ซับซ้อน
 //*********************************************************************
#define a 8                              //กำหนดให้ขา 8 ขับเซ็กเมนต์ a
#define b 7                              //กำหนดให้ขา 7 ขับเซ็กเมนต์ b
#define c 6                              //กำหนดให้ขา 6 ขับเซ็กเมนต์ c
#define d 5                              //กำหนดให้ขา 5 ขับเซ็กเมนต์ d
#define e 4                              //กำหนดให้ขา 4 ขับเซ็กเมนต์ e
#define f 3                              //กำหนดให้ขา 3 ขับเซ็กเมนต์ f
#define g 2                              //กำหนดให้ขา 2 ขับเซ็กเมนต์ g
unsigned int cnt = 0;                    //ประกาศตัวแปรชนิดเลขจำนวน cnt เต็มเพื่อใช้นับ
unsigned char pin[]={2,3,4,5,6,7,8};     //ประกาศตัวแปรอาเรย์ ตำแหน่งขาที่จะใช้

void setup() {                   //เริ่มฟังก์ชัน loop
 for(int  I = 0; I < 7; i++){    //วนลูปนับ 0-7
  pinMode(pin[i],OUTPUT);        //กำหนดให้ขาทั้ง 7 ทำงานเป็นเอาต์พุต
 }
 disp7(cnt);                     //เรียกใช้ฟังก์ชัน disp7 ให้แสดงค่า cnt คือ แสดงตัวเลข 0
 delay(1000);                    //หน่วงเวลา 1 วินาที
}                                //จบฟังก์ชัน loop

void loop() {                    //เริ่มฟังก์ชัน loop
 cnt++;                          //เพิ่มค่าตัวแปร cnt ขึ้น 1 ค่า
 if(cnt > 9)cnt=0;               //ถ้าหากค่าตัวแปร cnt มากกว่า 9 ให้ cnt มีค่าเป็น 0
 disp7(cnt);                     //เรียกใช้ฟังก์ชัน disp7 โดยส่งค่า cnt เข้าไปในฟังก์ชันด้วย
 delay(1000);                    //หน่วงเวลา 1 วินาที
}                                //จบฟังก์ชัน loop
     //ค่าของตัวแปร cnt จะถูกส่งเข้าฟังก์ชัน disp7 โดยส่งค่าข้อมูลผ่านตัวแปร num
void disp7(unsigned int num){            //เริ่มฟังก์ชัน disp7 เพื่อใช้ขับ led 7segment
 switch(num){                            //เช็คค่าตัวแปร num ที่รับเข้ามา
  case 0: {                              //หาก num มีค่าเป็น 0
   digitalWrite(a,HIGH);                 //ให้เซ็กเมนต์ a ติด
   digitalWrite(b,HIGH);                 //ให้เซ็กเมนต์ b ติด
   digitalWrite(c,HIGH);                 //ให้เซ็กเมนต์ c ติด
   digitalWrite(d,HIGH);                 //ให้เซ็กเมนต์ d ติด
```

```
  digitalWrite(e,HIGH);                //ให้เซ็กเมนต์ e ติด
  digitalWrite(f,HIGH);                //ให้เซ็กเมนต์ f ติด
  digitalWrite(g,LOW);                 //ให้เซ็กเมนต์ g ดับ__ทั้งหมดนี้เพื่อแสดงเลข 0
}break;
   case 1: {                           //หาก num มีค่าเป็น 1
  digitalWrite(a,LOW);                 //ให้เซ็กเมนต์ a ดับ
  digitalWrite(b,HIGH);                //ให้เซ็กเมนต์ b ติด
  digitalWrite(c,HIGH);                //ให้เซ็กเมนต์ c ติด
  digitalWrite(d,LOW);                 //ให้เซ็กเมนต์ d ดับ
  digitalWrite(e,LOW);                 //ให้เซ็กเมนต์ e ดับ
  digitalWrite(f,LOW);                 //ให้เซ็กเมนต์ f ดับ
  digitalWrite(g,LOW);                 //ให้เซ็กเมนต์ g ดับ__ทั้งหมดนี้เพื่อแสดงเลข 1
}break;
   case 2: {                           //หาก num มีค่าเป็น 2
  digitalWrite(a,HIGH);                //ให้เซ็กเมนต์ a ติด
  digitalWrite(b,HIGH);                //ให้เซ็กเมนต์ b ติด
  digitalWrite(c,LOW);                 //ให้เซ็กเมนต์ c ดับ
  digitalWrite(d,HIGH);                //ให้เซ็กเมนต์ d ติด
  digitalWrite(e,HIGH);                //ให้เซ็กเมนต์ e ติด
  digitalWrite(f,LOW);                 //ให้เซ็กเมนต์ f ดับ
  digitalWrite(g,HIGH);                //ให้เซ็กเมนต์ g ติด__ทั้งหมดนี้เพื่อแสดงเลข 2
}break;
   case 3: {                           //หาก num มีค่าเป็น 3
  digitalWrite(a,HIGH);                //ให้เซ็กเมนต์ a ติด
  digitalWrite(b,HIGH);                //ให้เซ็กเมนต์ b ติด
  digitalWrite(c,HIGH);                //ให้เซ็กเมนต์ c ติด
  digitalWrite(d,HIGH);                //ให้เซ็กเมนต์ d ติด
  digitalWrite(e,LOW);                 //ให้เซ็กเมนต์ e ดับ
  digitalWrite(f,LOW);                 //ให้เซ็กเมนต์ f ดับ
  digitalWrite(g,HIGH);                //ให้เซ็กเมนต์ g ติด__ทั้งหมดนี้เพื่อแสดงเลข 3
}break;
   case 4:{                            //หาก num มีค่าเป็น 4
  digitalWrite(a,LOW);                 //ให้เซ็กเมนต์ a ดับ
  digitalWrite(b,HIGH);                //ให้เซ็กเมนต์ b ติด
  digitalWrite(c,HIGH);                //ให้เซ็กเมนต์ c ติด
  digitalWrite(d,LOW);                 //ให้เซ็กเมนต์ d ดับ
```

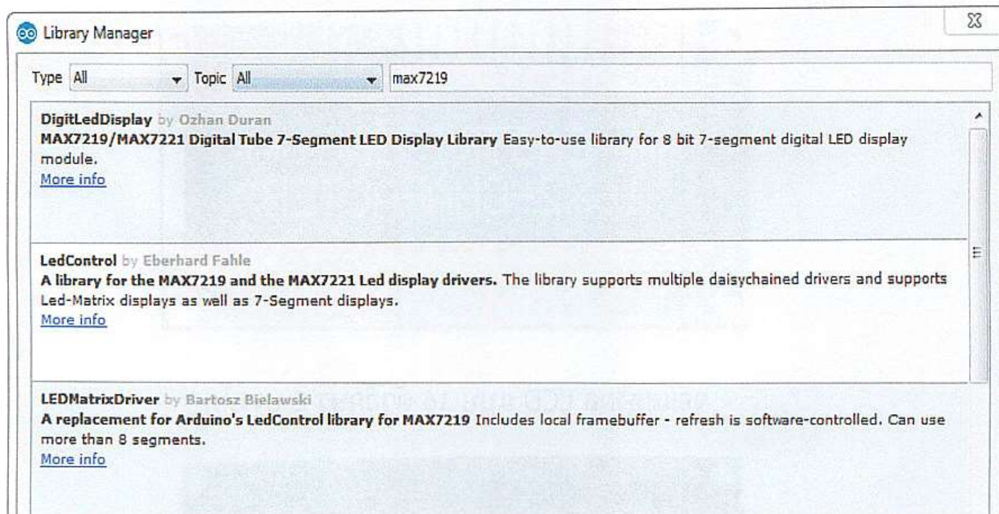
```
    digitalWrite(e,LOW);          //ให้เซ็กเมนต์ e ดับ
    digitalWrite(f,HIGH);         //ให้เซ็กเมนต์ f ติด
    digitalWrite(g,HIGH);         //ให้เซ็กเมนต์ g ติด__ทั้งหมดนี้เพื่อแสดงเลข 4
  }break;
    case 5: {                     //หาก num มีค่าเป็น 5
    digitalWrite(a,HIGH);         //ให้เซ็กเมนต์ a ติด
    digitalWrite(b,LOW);          //ให้เซ็กเมนต์ b ดับ
    digitalWrite(c,HIGH);         //ให้เซ็กเมนต์ c ติด
    digitalWrite(d,HIGH);         //ให้เซ็กเมนต์ d ติด
    digitalWrite(e,LOW);          //ให้เซ็กเมนต์ e ดับ
    digitalWrite(f,HIGH);         //ให้เซ็กเมนต์ f ติด
    digitalWrite(g,HIGH);         //ให้เซ็กเมนต์ g ติด__ทั้งหมดนี้เพื่อแสดงเลข 5
  }break;
    case 6: {                     //หาก num มีค่าเป็น 6
    digitalWrite(a,HIGH);         //ให้เซ็กเมนต์ a ติด
    digitalWrite(b,LOW);          //ให้เซ็กเมนต์ b ดับ
    digitalWrite(c,HIGH);         //ให้เซ็กเมนต์ c ติด
    digitalWrite(d,HIGH);         //ให้เซ็กเมนต์ d ติด
    digitalWrite(e,HIGH);         //ให้เซ็กเมนต์ e ติด
    digitalWrite(f,HIGH);         //ให้เซ็กเมนต์ f ติด
    digitalWrite(g,HIGH);         //ให้เซ็กเมนต์ g ติด__ทั้งหมดนี้เพื่อแสดงเลข 6
  }break;
    case 7: {                     //หาก num มีค่าเป็น 7
    digitalWrite(a,HIGH);         //ให้เซ็กเมนต์ a ติด
    digitalWrite(b,HIGH);         //ให้เซ็กเมนต์ b ติด
    digitalWrite(c,HIGH);         //ให้เซ็กเมนต์ c ติด
    digitalWrite(d,LOW);          //ให้เซ็กเมนต์ d ดับ
    digitalWrite(e,LOW);          //ให้เซ็กเมนต์ e ดับ
    digitalWrite(f,LOW);          //ให้เซ็กเมนต์ f ดับ
    digitalWrite(g,LOW);          //ให้เซ็กเมนต์ g ดับ__ทั้งหมดนี้เพื่อแสดงเลข 7
  }break;
    case 8: {                     //หาก num มีค่าเป็น 8
    digitalWrite(a,HIGH);         //ให้เซ็กเมนต์ a ติด
    digitalWrite(b,HIGH);         //ให้เซ็กเมนต์ b ติด
    digitalWrite(c,HIGH);         //ให้เซ็กเมนต์ c ติด
    digitalWrite(d,HIGH);         //ให้เซ็กเมนต์ d ติด
```

```
    digitalWrite(e,HIGH);           //ให้เซ็กเมนต์ e ติด
    digitalWrite(f,HIGH);           //ให้เซ็กเมนต์ f ติด
    digitalWrite(g,HIGH);           //ให้เซ็กเมนต์ g ติด__ทั้งหมดนี้เพื่อแสดงเลข 8
  }break;
     case 9: {                      //หาก num มีค่าเป็น 9
    digitalWrite(a,HIGH);           //ให้เซ็กเมนต์ a ติด
    digitalWrite(b,HIGH);           //ให้เซ็กเมนต์ b ติด
    digitalWrite(c,HIGH);           //ให้เซ็กเมนต์ c ติด
    digitalWrite(d,HIGH);           //ให้เซ็กเมนต์ d ติด
    digitalWrite(e,LOW);            //ให้เซ็กเมนต์ e ดับ
    digitalWrite(f,HIGH);           //ให้เซ็กเมนต์ f ติด
    digitalWrite(g,HIGH);           //ให้เซ็กเมนต์ g ติด__ทั้งหมดนี้เพื่อแสดงเลข 9
  }break;
 }
}
```

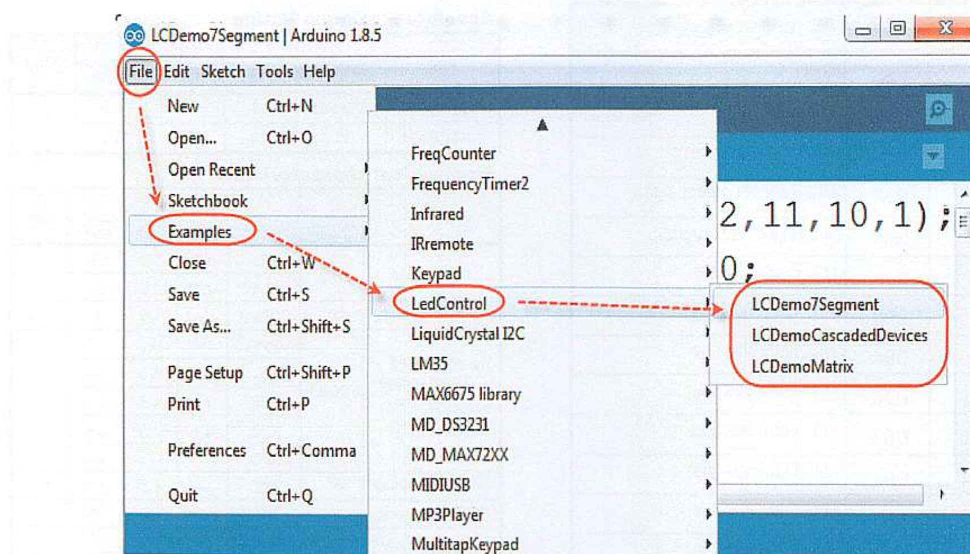
โมดูล LED 7 Segment สำเร็จรูป MAX7219

โมดูลขับ led 7 segment แบบสำเร็จรูปที่ใช้ไอซี MAX7219 ทำให้การแสดงผลด้วย led 7 segment นั้นง่ายขึ้น เพราะนอกจากจะมาในรูปแบบของวงจรสำเร็จรูปที่พร้อมต่อใช้งานได้อย่างง่ายดายแล้ว ยังมีไลบรารีสำเร็จรูปให้ดาวน์โหลดมาใช้งานได้ทันที และมีไลบรารีหลากหลายรูปแบบ





สำหรับการใช้งานโมดูลสำเร็จรูปนี้ผู้เขียนจะไม่ระบุการใช้เฉพาะไลบรารีแบบใดแบบหนึ่ง แต่จะขอกล่าวกว้างๆให้ครอบคลุมทั้งหมด เนื่องจากไลบรารีมีหลายแบบ เพราะมีผู้พัฒนาหลายคน แล้วแต่ผู้ใช้งานจะเลือกโหลดตัวไหนมาใช้งาน เมื่อดาวน์โหลดไลบรารีมาติดตั้งเรียบร้อยแล้ว ผู้ใช้สามารถเรียกดูตัวอย่างการใช้งานได้จาก เมนู File → Examples ก็จะได้ตัวอย่างการใช้งาน และ ให้จำเอาไว้ว่า ไม่ใช่เฉพาะแต่ไลบรารี led 7 segment เท่านั้น ไม่ว่าจะเป็นไลบรารีสำเร็จรูปของอะไรก็ตาม จะมีตัวอย่างการใช้งานมาให้ทั้งหมด ผู้ใช้งานสามารถเปิดดูตัวอย่างและนำไปใช้ได้อย่างง่ายดาย ด้วยวิธีการเดียวกันนี้



จอแสดงผล LCD

จอแสดงผลอีกแบบหนึ่งที่นิยมใช้กันมากคือจอแสดงผลแบบผลึกเหลวหรือ เรียกว่า จอ LCD หนังสือเล่มนี้จะไม่เขียนลึกลงไปถึงโครงสร้างและการทำงานระดับลึกแต่จะเน้นไปที่การใช้งานที่ง่าย ที่ใครๆก็สามารถนำมาใช้งานได้ ด้วยการดาวน์โหลดและใช้งานไลบรารีสำเร็จรูป ที่ Arduino มีให้



จอแสดงผล LCD แบบ 16 ตัวอักษร 2 บรรทัด



จอแสดงผล LCD แบบ 20 ตัวอักษร 4 บรรทัด

PIN NO	Symbol	Fuction
1	VSS	GND
2	VDD	+5V
3	V0	Contrast adjustment
4	RS	H/L Register select signal
5	R/W	H/L Read/Write signal
6	E	H/L Enable signal
7	DB0	H/L Data bus line
8	DB1	H/L Data bus line
9	DB2	H/L Data bus line
10	DB3	H/L Data bus line
11	DB4	H/L Data bus line
12	DB5	H/L Data bus line
13	DB6	H/L Data bus line
14	DB7	H/L Data bus line
15	A	+4.2V for LED
16	K	Power supply for BKL(0V)

Absolute Maximum Rating

Item	Symbol	Standard			Unit
		Min	Typ	Max	
Power supply	*VDD-VSS*	-0.3	--------	5.5	V
Input voltage	*Vi*	-0.3	--------	VDD	

Electronical characteristics

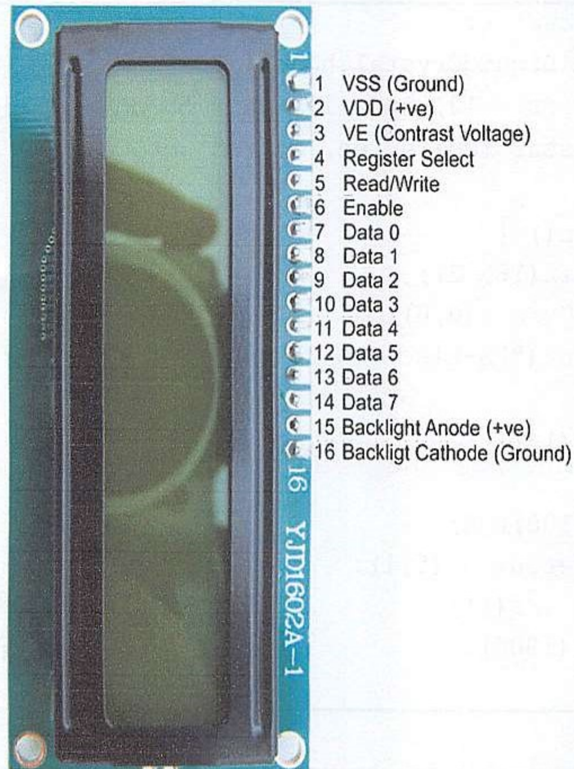
Item	Symbol	Condition	Standard			Unit
			Min	Typ	Max	
Input voltage	*VDD*	+5V	4.7	5.0	5.5	V
		+3.3V	2.7	3.0	5.3	V
Supply current	*IDD*	VDD=5V	------	1.5	4	mA
Recommended LCD riling voltage for normal temp version module	VDD-V0	-20°C	------	------	------	V
		0 °C	4.7	5.0	5.5	
		25°C	4.3	4.5	4.7	
		50°C	4.1	4.3	4.5	
		70°C	------	-------	------	
LED forward voltage	*VF*	25°C	------	4.2	4.6	V
LED forward current	*IF*	25°C	------	120	160	mA
EL power supply current	IEL	V_{EL}=110V AC 400Hz	-----	-------	------	mA

Display character address code:

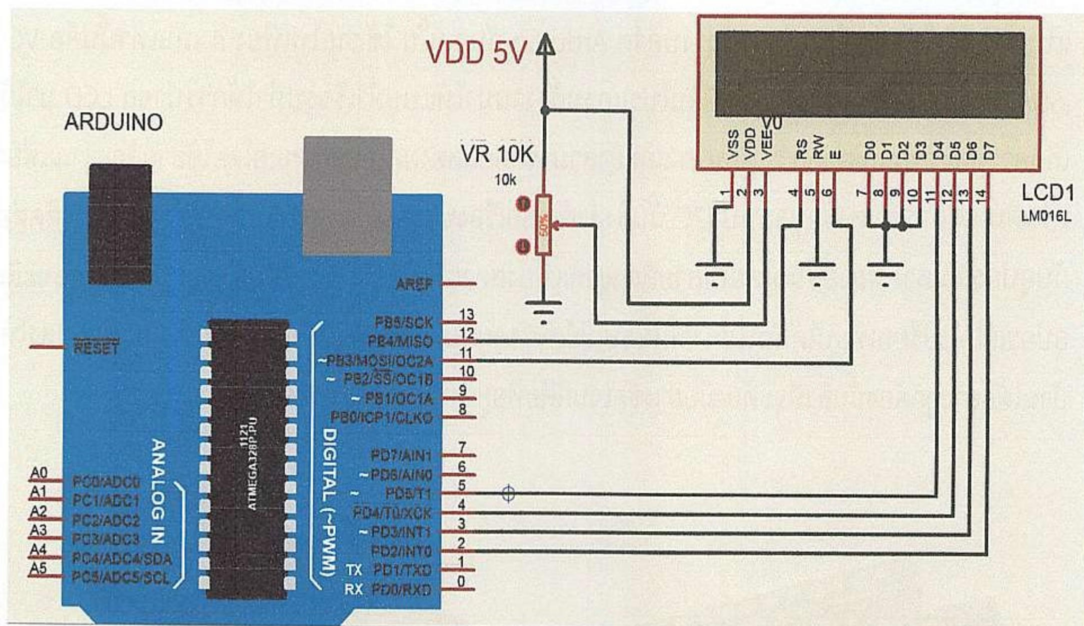
Display position

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
DDRAM address	00	01	02	---	---	---	---	---								0FH
DDRAM address	40	41	42	---	---	---	---	---								4FH

ภาพแสดงตำแหน่งการต่อใช้งานขาต่างๆ



ทดสอบโปรแกรมแสดงผลด้วยจอ LCD 16*2

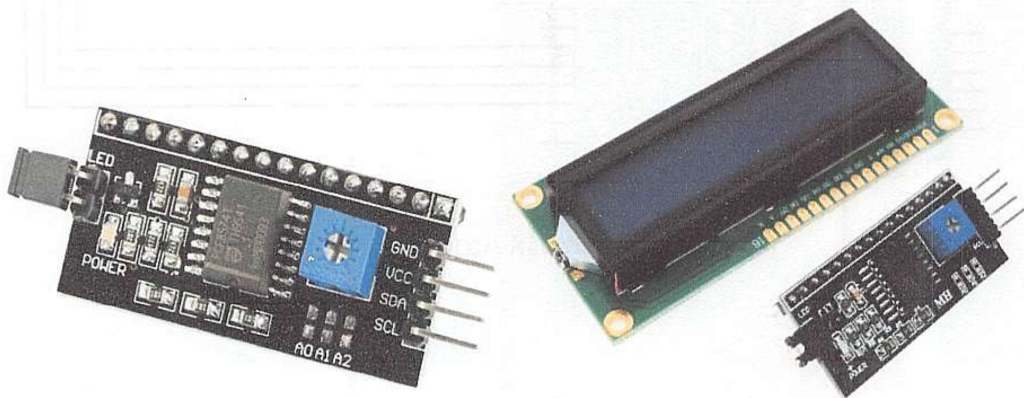


การต่อวงจรใช้งานบอร์ด Arduino เข้ากับจอ LCD 16*2

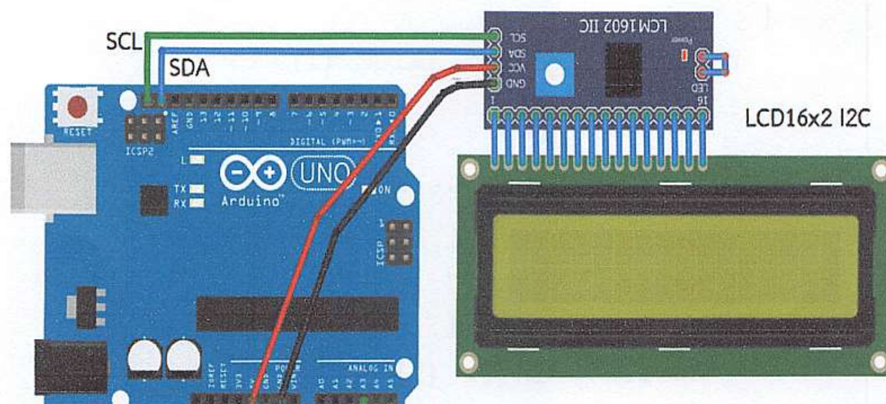
```
unsigned char t;
#include <LiquidCrystal.h>
const int rs = 12, en = 11, d4 = 5, d5 = 4, d6 = 3, d7 = 2;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);
int cnt;
void setup() {
  lcd.begin(16, 2);
  lcd.setCursor(0,0);
  lcd.print("PS-Electronics");
}
void loop() {
    t++;
    if(t>100)t=0;
    lcd.setCursor(5,1);
    lcd.print(t);
    delay(1000);
}
```

โปรแกรมนับ 0-100 แสดงผลด้วยจอ LCD 16*2

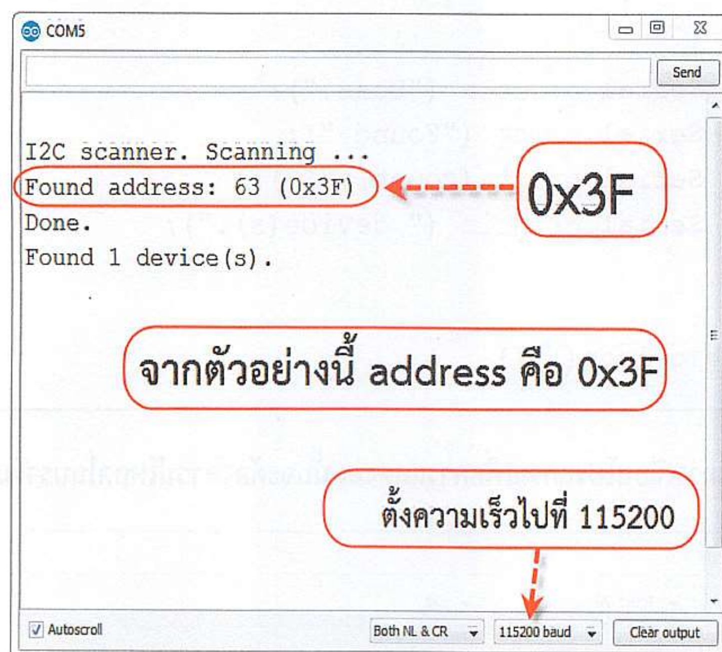
ถึงแม้ว่าการเขียนโปรแกรมแสดงผลด้วยจอ LCD นั้นจะมีความสะดวกเนื่องจากมีไลบรารีสำเร็จรูปมาให้ แต่การต่อใช้งานนั้นค่อนข้างยุ่งยากเพราะต้องต่อสายไฟจำนวนหลายเส้นเพื่อใช้งาน และตัวจอเองก็ต้องมีการต่อ VR เพื่อปรับความเข้มของการแสดงผลตัวอักษร แต่ปัจจุบันก็ได้มีโมดูลขับจอแสดงผล LCD เป็นโมดูลสำเร็จรูปสามารถต่อเข้ากับจอ LCD ได้อย่างง่ายด้วยและการเขียนโปรแกรมสั่งงานนั้นก็ใช้การรับ-ส่งข้อมูลแบบ I2C ทำให้การเชื่อมต่อระหว่างบอร์ด Arduino กับจอนั้น ใช้สายไฟเพียง 4 เส้นเท่านั้นคือ VCC ,GND ,SDA, SCL และมีไลบรารีในการเขียนโปรแกรมสั่งงานมาให้แบบสำเร็จรูปทำให้การนำจอ LCD มาใช้งานนั้นเกิดความสะดวกและง่ายกว่าเดิมมาก และยังสามารถต่อแสดงผลได้หลายๆจอโดยใช้สายสัญญาณเพียง 4 เส้นเท่าเดิม แต่การรับ-ส่งข้อมูลแบบ I2C นั้นสิ่งสำคัญคือเรื่องของแอดเดรสของอุปกรณ์ เพราะการติดต่อสื่อสารกับอุปกรณ์หลากหลายชนิดโดยใช้สายสัญญาณเพียงสองเส้นนั้น จะต้องมีตัวบ่งบอกว่าต้องการจะติดต่อกับอุปกรณ์ใด เปรียบเทียบได้กับพนักงานไปรษณีย์เพียงคนเดียว การที่จะนำจดหมายหลายฉบับไปส่งให้หลายๆบ้านได้อย่างถูกต้องนั้นสิ่งที่จะต้องมีอย่างชัดเจนก็คือที่อยู่ หรือ แอดเดรส (Address) นั่นเอง



ที่อยู่ หรือ Address ของบุคคลนั้นคือ ชื่อ บ้านเลขที่ ตำบล อำเภอ จังหวัด และ รหัสไปรษณีย์ แต่ที่อยู่ หรือ Address ของอุปกรณ์ที่ติดต่อสื่อสารกันแบบ I2C นั้นเป็นรหัสตัวเลขฐาน 16 ซึ่ง ในที่นี้จะกล่าวถึงเฉพาะโมดูลขับจอ LCD แบบ I2C เท่านั้นซึ่งโมดูลที่มีขายทั่วไปที่ได้รับความนิยมนั้นใช้ไอซีเบอร์ PCF8574 และ PCF8574T ซึ่งไอซีทั้งสองเบอร์นี้มี Address ไม่เหมือนกัน และบนตัวโมดูลเองก็ยังมีจุดจั๊มเปอร์มาให้สำหรับให้ผู้ใช้สามารถจะกำหนด Address ขึ้นมาใช้เองได้อีกด้วย โดยต้องดูข้อมูลจากดาต้าชีท เรื่องนี้มีปัญหามากสำหรับผู้เริ่มต้นใช้งานที่ซื้อโมดูลมาใช้โดยที่ยังไม่มีความรู้ในเรื่องนี้ เมื่อเขียนโค้ดตามตัวอย่างก็จะไม่สามารถแสดงผลออกมาได้ และในหนังสือเล่มนี้จะมีโปรแกรมที่ใช้ในการสแกน address ของจอให้ด้วย ขั้นตอนการสแกนหาแอดเดรสนั้นให้ต่อจอเข้ากับบอร์ด Arduino ตามรูป โดยเมื่ออัพโหลดโปรแกรมแล้ว ให้ทำการต่อจอเข้ากับบอร์ด Arduino แล้วกดดูข้อมูลได้ที่ Serial Monitor



การต่อวงจรใช้งาน LCD แบบ I2C



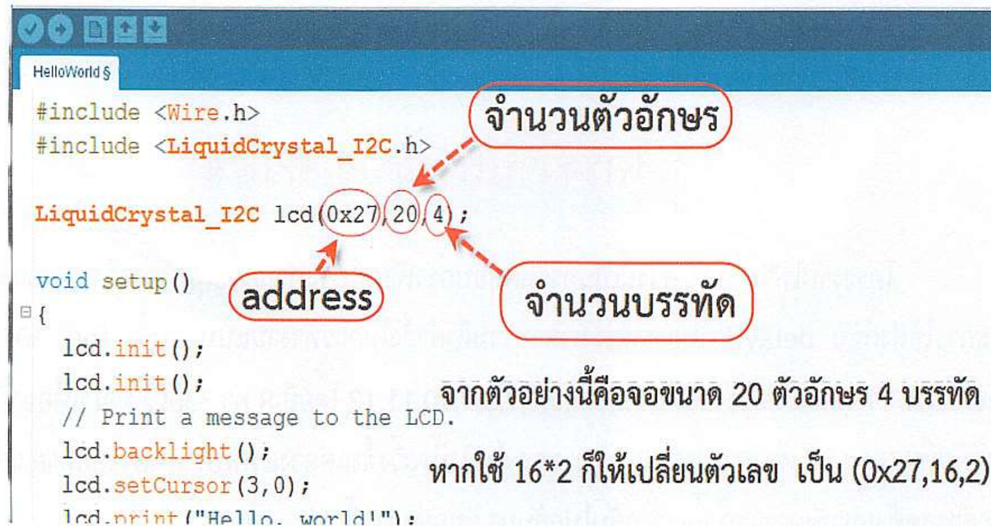
โปรแกรมที่ใช้ในการสแกน address

```
#include <Wire.h>
void setup() {
  Serial.begin (115200);
  while (!Serial)
    {
    }
  Serial.println ();
  Serial.println ("I2C scanner. Scanning ...");
  byte count = 0;
  Wire.begin();
  for (byte i = 1; i < 120; i++)
  {
    Wire.beginTransmission (i);
    if (Wire.endTransmission () == 0)
      {
      Serial.print ("Found address: ");
      Serial.print (i, DEC);
      Serial.print (" (0x");
      Serial.print (i, HEX);
      Serial.println (")");
      count++;
      delay (1);
      }
  }
  Serial.println ("Done.");
  Serial.print ("Found ");
  Serial.print (count, DEC);
  Serial.println (" device(s).");
}

void loop() {}
```

ส่วนการเขียนโปรแกรมเพื่อการแสดงผลนั้นจะต้องดาวน์โหลดไลบรารีมาติดตั้งก่อน





ส่วนตัวเลข 0x27 อาจเป็นเลขอื่น เช่น 0x3F ขึ้นอยู่กับ address ที่ใช้ ให้ใส่ตามจริง

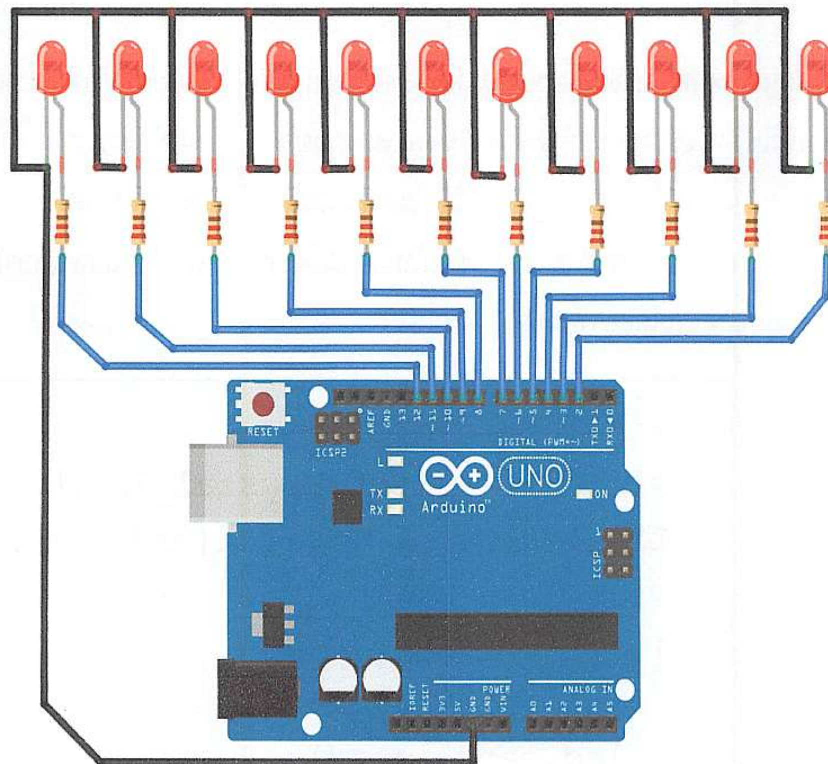
และขั้นตอนการเขียนโปรแกรมเพื่อแสดงผลนั้นจะต้องใส่ address และขนาดการแสดงผลของจอให้ตรงกับจอที่ใช้งานจริงจากรูปตัวอย่างเป็นจอแสดงผลขนาด 20*4 คือสามารถแสดงผลตัวอักษรได้ 20 ตัวอักษร จำนวน 4 บรรทัด แต่หากผู้ใช้งานใช้จอแสดงผลขนาด 16*2 นั่นหมายความว่ามันแสดงผลได้เพียง 16 ตัวอักษร จำนวน 2 บรรทัด ดังนั้นเวลาเขียนโปรแกรมใช้งาน จะต้องกำหนดค่าเหล่านี้ให้ตรงกับความเป็นจริงจึงจะสามารถแสดงผลได้อย่างถูกต้อง

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x3F,16,2);
void setup()
{
  lcd.init();
  lcd.backlight();
  lcd.setCursor(4,0);
  lcd.print("Arduino");
  lcd.setCursor(0,1);
  lcd.print("My firs project");

}
void loop(){}
```

โครงงานไฟวิ่ง 11 ดวง

โครงงานไฟวิ่ง 11 ดวงนี้เป็นการทดสอบการเขียนโปรแกรมควบคุม การสั่งงานเอาต์พุตเบื้องต้น และการใช้ฟังก์ชัน **delay()** ในการหน่วงเวลา รวมถึงคำสั่งในการทำงานแบบ วนซ้ำ **for()** โครงงานนี้จะใช้หลอด LED 11 หลอด ต่อใช้งานที่ขา **2,3,4,5,6,7,8,9,10,11,12** โดยมี R ค่า 330**Ω** ต่อไว้เพื่อจำกัดกระแสให้พอดีกับที่หลอด LED ต้องการ และการทำงานของไฟวิ่งนี้จะวิ่งตั้งแต่ดวงแรกไปจนถึงดวงสุดท้าย แล้วจะวิ่งกลับจากดวงสุดท้ายมายังดวงแรก และวิ่งกลับไปกลับมา วนอยู่อย่างนี้

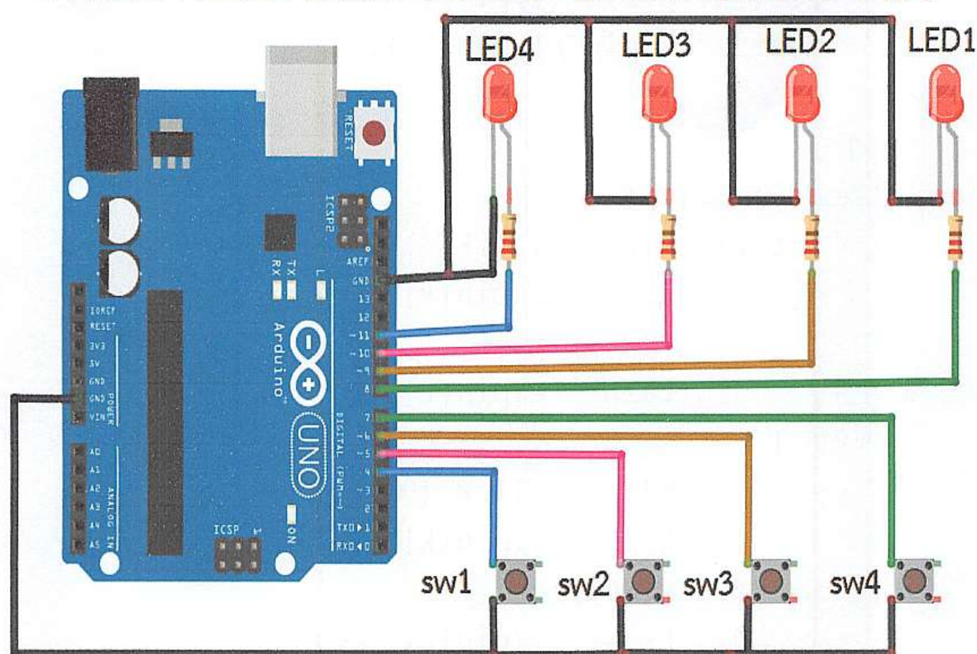


```
unsigned char pin[] = {2,3,4,5,6,7,8,9,10,11,12};
void setup() {
  for(int p=0;p<11;p++){
    pinMode(pin[p],OUTPUT);
  }
}
void loop() {
    for(int k=0;k<11;k++){
      digitalWrite(pin[k],HIGH);
      delay(200);
      digitalWrite(pin[k],LOW);
    }
        for(int k=11;k>(0-1);k--){
      digitalWrite(pin[k],HIGH);
      delay(200);
      digitalWrite(pin[k],LOW);
    }
}
```

โค้ดโปรแกรมไฟวิ่ง 11 ดวง วิ่งไปกลับโดยใช้ลูป for ในการนับค่าตัวแปร

โปรแกรมสั่งงาน เปิด-ปิด led 4 ดวง ด้วยสวิตซ์ 4 ตัว แยกอิสระ

การต่อวงจรสั่งงานเปิด-ปิด LED 4ดวง ด้วยสวิตซ์ 4 ตัว



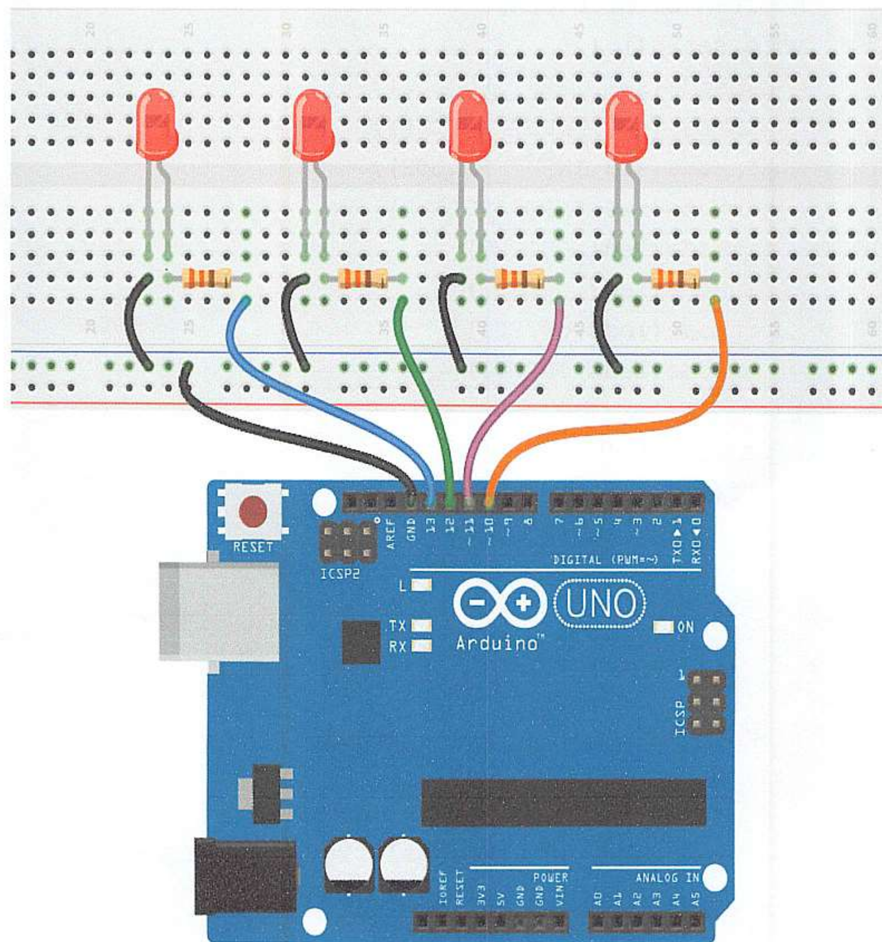
โปรแกรมสวิตซ์เปิด-ปิดไฟนี้เราจะใช้ฟังก์ชัน **digitalRead()** เพื่ออ่านค่าของพอร์ตที่ต่ออยู่กับสวิตซ์แต่ละตัวคือ ขา 4,5,6 และ 7 โดยการประกาศใช้งานเป็นอินพุต และเปิดใช้งาน R พูลอัพที่อยู่ภายในด้วย จึงส่งผลให้ขา 4,5,6,7 มีสถานะเป็น HIGH หรือมีไฟ เมื่อมีการกดสวิตซ์จะเป็นการต่อขานั้นลงกราวด์ไป จึงทำให้ขานั้นมีสถานะเป็น LOW คือไม่มีไฟ ด้วยสถานะที่เปลี่ยนแปลงได้แค่ 2 สถานะ คือ มีไฟ กับ ไม่มีไฟ และเราจะใช้สวิตซ์เพียงตัวเดียวในการสั่งงานเพื่อให้เปิดไฟ และ ปิดไฟได้ด้วย จึงใช้ตัวแปรชนิด boolean เข้ามาใช้งาน เมื่อมีการกดสวิตซ์จะสั่งให้กลับสถานะของตัวแปร boolean ทุกครั้งเมื่อมีการกดสวิตซ์ คือกดครั้งแรกจะเปลี่ยนค่าจาก false ให้เป็น true และหากกดสวิตซ์อีกครั้ง จะเปลี่ยนจาก true ให้เป็น false และจะวนอยู่อย่างนี้ไปเรื่อยๆเพราะ boolean นั้นมีค่าที่เปลี่ยนได้แค่สองค่านี้เท่านั้น ดังนั้นในการกดสวิตซ์แต่ละครั้ง ค่าตัวแปร boolean ก็จะเปลี่ยนกลับไปมาระหว่าง true กับ false แล้วเราจึงใช้คำสั่ง **if** เพื่อตรวจสอบค่าตัวแปร boolean นี้ หากค่าเป็น true ก็เขียนโปรแกรมสั่งให้หลอดไฟติด และหากค่าเป็น false ก็เขียนโปรแกรมสั่งให้หลอดไฟดับ ดังนั้นเราจึงสามารถสั่งให้หลอด LED ติด และ ดับ ได้ด้วยสวิตซ์ button เพียงตัวเดียวนั่นเอง

```
#define sw1 4
#define sw2 5
#define sw3 6
#define sw4 7
#define led1 8
#define led2 9
#define led3 10
#define led4 11
boolean b1,b2,b3,b4;
void setup() {
  pinMode(sw1,INPUT_PULLUP);
  pinMode(sw2,INPUT_PULLUP);
  pinMode(sw3,INPUT_PULLUP);
  pinMode(sw4,INPUT_PULLUP);
  pinMode(led1,OUTPUT);
  pinMode(led2,OUTPUT);
  pinMode(led3,OUTPUT);
  pinMode(led4,OUTPUT);
}
void loop() {
   if(digitalRead(sw1)==0){
      delay(200);
      b1 = !b1;
      if(b1==true) digitalWrite(led1,HIGH);
      else digitalWrite(led1,LOW);
    }
   else if(digitalRead(sw2)==0){
       delay(200);
       b2 = !b2;
       if(b2==true) digitalWrite(led2,HIGH);
       else digitalWrite(led2,LOW);
   }
   else if(digitalRead(sw3)==0){
       delay(200);
       b3 = !b3;
       if(b3==true) digitalWrite(led3,HIGH);
       else digitalWrite(led3,LOW);
   }
   else if(digitalRead(sw4)==0){
       delay(200);
       b4 = !b4;
       if(b4==true) digitalWrite(led4,HIGH);
       else digitalWrite(led4,LOW);
   }
}
```

ตัวอย่างการใช้งาน การรับ-ส่งข้อมูล Serial UART

การรับ-ส่งข้อมูลกับคอมพิวเตอร์สั่งเปิด-ปิดไฟ

การต่อวงจรใช้ในการทดลอง ในบอร์ดทดลอง



โครงงานนี้เราจะมาดูวิธีการติดต่อสื่อสารระหว่างคอมพิวเตอร์กับบอร์ด Arduino เพราะเราจะพิมพ์ข้อความคำสั่งจาก Serial Monitor เพื่อสั่งงานให้บอร์ด Arduino เปิดไฟ และปิดไฟได้ และโครงงานนี้ไม่มีอะไรยาก และไม่ต้องใช้โปรแกรมคอมพิวเตอร์มาสั่งงาน เพราะเราจะพิมพ์คำสั่งด้วย Serial Monitor ที่มากับ Arduino อยู่แล้ว

หลักการทำงานก็ไม่ได้มีอะไรซับซ้อนเราเพียงแค่ประกาศตัวแปรขึ้นมารับข้อมูลที่ส่งมาจากคอมพิวเตอร์โดยการเขียนคำสั่งไว้ว่าหากพบข้อมูลที่เป็นการสั่งให้เปิดไฟ ก็ให้สั่งเปิดไฟ โดยข้อมูลที่ส่งมาจากคอมพิวเตอร์นั้นอาจจะเป็นตัวเลข หรือตัวอักษร หรือตัวเลขก็ได้ ทั้งนี้ขึ้นอยู่กับผู้สร้างโครงงานนี้ว่าจะเขียนโปรแกรมอย่างไร

แต่ผู้เขียนจะใช้คำสั่งเปิดไฟดวงที่หนึ่ง L1ON คำสั่งเปิดไฟดวงที่สอง L2ON คำสั่งเปิดไฟดวงที่สาม L3ON และคำสั่งเปิดไฟดวงที่สี่ L4ON และในส่วนของคำสั่งเพื่อสั่งให้ปิดไฟนั้นจะใช้คำสั่งคล้ายๆกับการสั่งเปิดไฟ เพียงแค่เปลี่ยนจาก ON เป็น OFF เท่านั้นเองเพื่อป้องกันความสับสนและให้เข้าใจได้โดยง่ายคำสั่งปิดไฟดวงที่หนึ่ง L1OFF คำสั่งปิดไฟดวงที่สอง L2OFF ปิดไฟดวงที่สาม L3OFF ปิดไฟดวงที่สี่ L4OFF คำสั่งเปิดและปิดไฟนี้ผู้เขียนโปรแกรมเป็นคนกำหนดขึ้นมาเองดังนั้นอยู่ที่ใครจะกำหนดคำหรือข้อความหรือตัวเลขเพื่อใช้ในการเปิดละปิดไฟกันเอาเอง นี่เป็นเพียงตัวอย่างเพื่อแสดงให้ผู้อ่านได้ทำความเข้าใจได้ง่ายๆเท่านั้น ผู้อ่านไม่จำเป็นต้องกำหนดข้อความตามหนังสือก็ได้

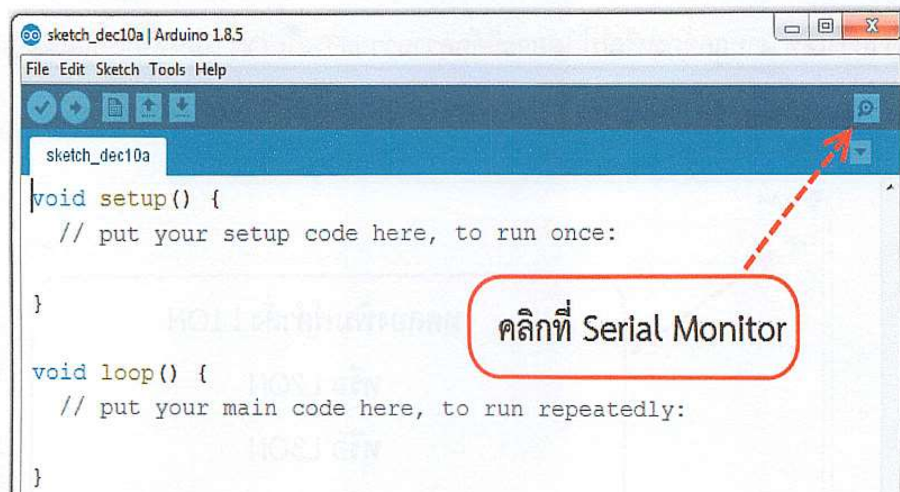
	คำสั่งเปิดไฟ	คำสั่งปิดไฟ
ไฟดวงที่1	L1ON	L1OFF
ไฟดวงที่2	L2ON	L2OFF
ไฟดวงที่3	L3ON	L3OFF
ไฟดวงที่4	L4ON	L4OFF

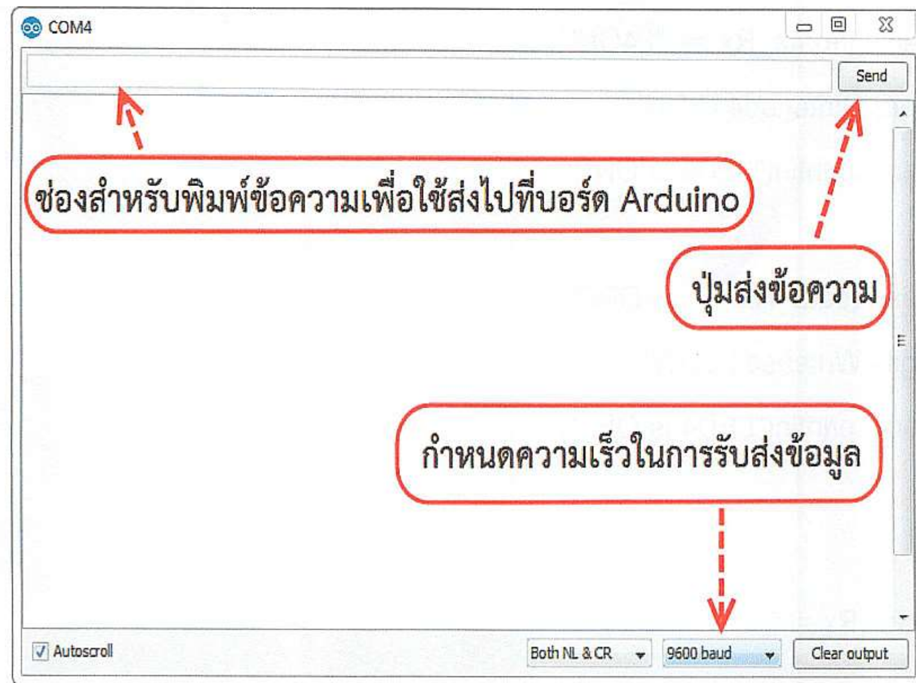
```
//____________โค้ดโปรแกรมสั่งงานเปิด-ปิดไฟจากคอมพิวเตอร์
#define led1 13
#define led2 12
#define led3 11
#define led4 10
String data_Rx;
void setup() {
 Serial.begin(9600);
 pinMode(led1,OUTPUT);
 pinMode(led2,OUTPUT);
 pinMode(led3,OUTPUT);
 pinMode(led4,OUTPUT);
}
```

```
void loop() {
 while(Serial.available()>0){
  delay(3);
  char rx_char = Serial.read();
  data_Rx += rx_char;
   if(data_Rx == "L1ON"){
  digitalWrite(led1,HIGH);
  Serial.println("LED1 is ON");
  }
  else if(data_Rx == "L1OFF"){
  digitalWrite(led1,LOW);
  Serial.println("LED1 is OFF");
 }
    else if(data_Rx == "L2ON"){
  digitalWrite(led2,HIGH);
  Serial.println("LED2 is ON");
    }
  else if(data_Rx == "L2OFF"){
  digitalWrite(led2,LOW);
  Serial.println("LED2 is OFF");
 }
    else if(data_Rx == "L3ON"){
  digitalWrite(led3,HIGH);
  Serial.println("LED3 is ON");
  }
  else if(data_Rx == "L3OFF"){
  digitalWrite(led3,LOW);
  Serial.println("LED2 is OFF");
 }
```

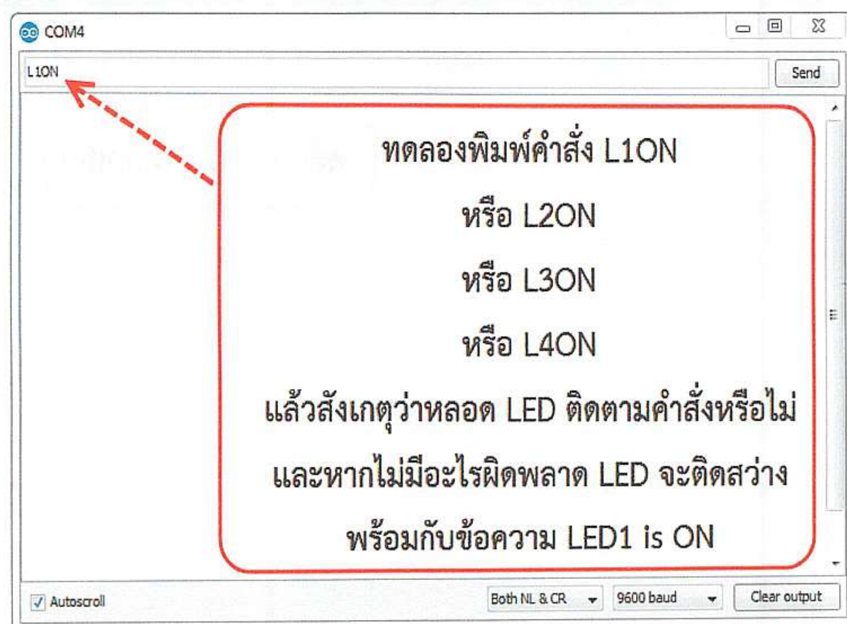
```
  else if(data_Rx == "L4ON"){
  digitalWrite(led4,HIGH);
  Serial.println("LED4 is ON");
  }
  else if(data_Rx == "L4OFF"){
  digitalWrite(led4,LOW);
  Serial.println("LED4 is OFF");
  }
 }
  data_Rx = "";
}
```

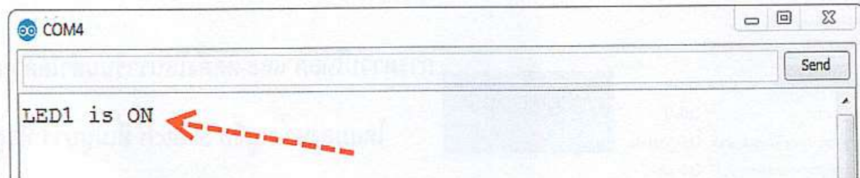
เมื่อเขียนโค้ดและได้อัพโหลดโปรแกรมลงบอร์ด Arduino แล้วให้คลิกเปิด Serial Monitor ขึ้นมา





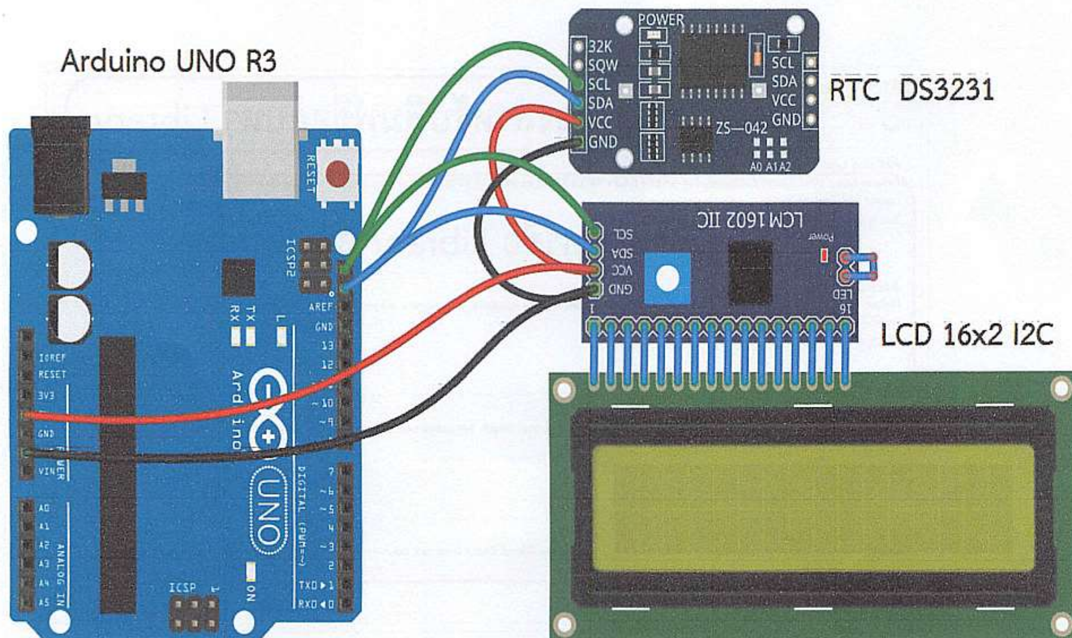
เมื่อเปิด Serial Monitor ขึ้นมาแล้วให้ทำการทดลองพิมพ์คำสั่งดูว่าสามารถสั่งงานให้หลอดติด และดับได้หรือไม่ หากไม่มีอะไรผิดพลาดจะสามารถสั่งงานควบคุมเปิดและปิดหลอดไฟได้ และเมื่อ Arduino รับคำสั่งเปิดหรือปิดไฟได้แล้ว หลังจากสั่งงานเปิดหรือปิดไฟเสร็จก็จะส่งข้อความกลับมาที่คอมพิวเตอร์ เพื่อแจ้งให้ทราบว่าได้ทำงานถูกต้องหรือไม่ โดยจะมีข้อความว่า LEDx is ON เนื่องจากเราได้เขียนคำสั่งเอาไว้ให้ส่งข้อความกลับมาที่คอมพิวเตอร์





เมื่อสั่งงานเสร็จก็จะส่งข้อความกลับมาที่คอมพิวเตอร์

การต่อใช้งานโมดูลนาฬิกา DS3231 กับบอร์ด Arduino UNO R3



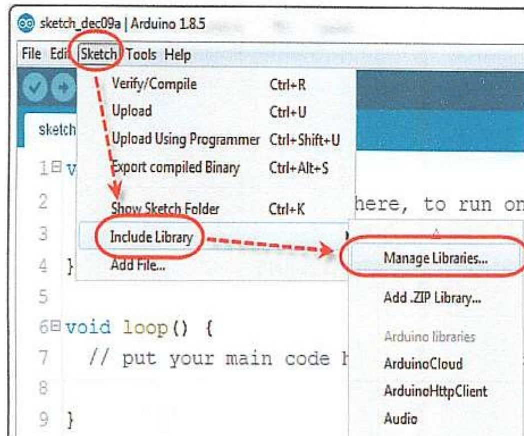
ตัวอย่างการใช้งาน Arduino กับเซ็นเซอร์ต่างๆ

ในการเขียนโปรแกรมใช้งาน Aruino เพื่อเชื่อมต่อกับเซ็นเซอร์ต่างๆนั้นเป็นเรื่องที่แสนง่ายดายกว่าไมโครคอนโทรลเลอร์ตระกูลอื่น เพราะว่า Arduino นั้นมีไบรารีสำเร็จรูปไว้ให้ใช้งานได้เพียงแค่ดาวน์โหลดโหลดไลบรารีของเซ็นเซอร์นั้นมา ก็จะมีโค้ดตัวอย่างการใช้งานอย่างครบถ้วน

การค้นหาและติดตั้งไลบรารีของเซ็นเซอร์ต่างๆ

ในการค้นหาและติดตั้งไลบรารีนั้น จะต้องมีการเชื่อมต่ออินเตอร์เน็ตไว้ด้วย

ตัวอย่างการใช้งานเซ็นเซอร์วัดอุณหภูมิ LM35

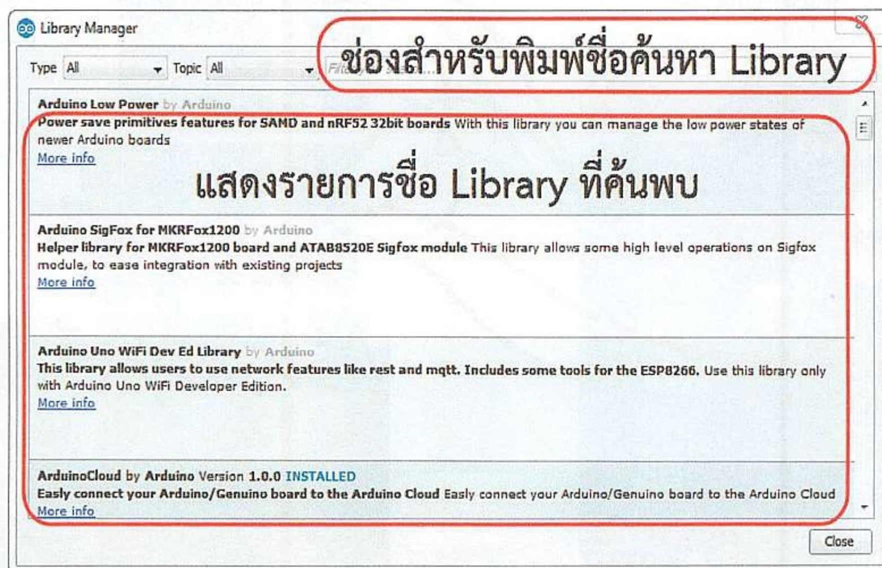


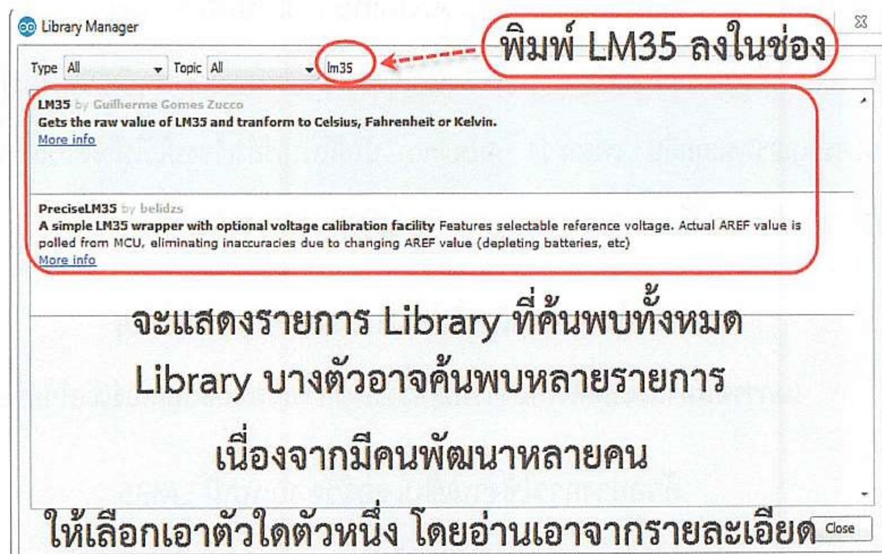
การดาวน์โหลด และ ติดตั้งไลบรารีนั้นทำได้ด้วยขั้นตอนง่ายๆ

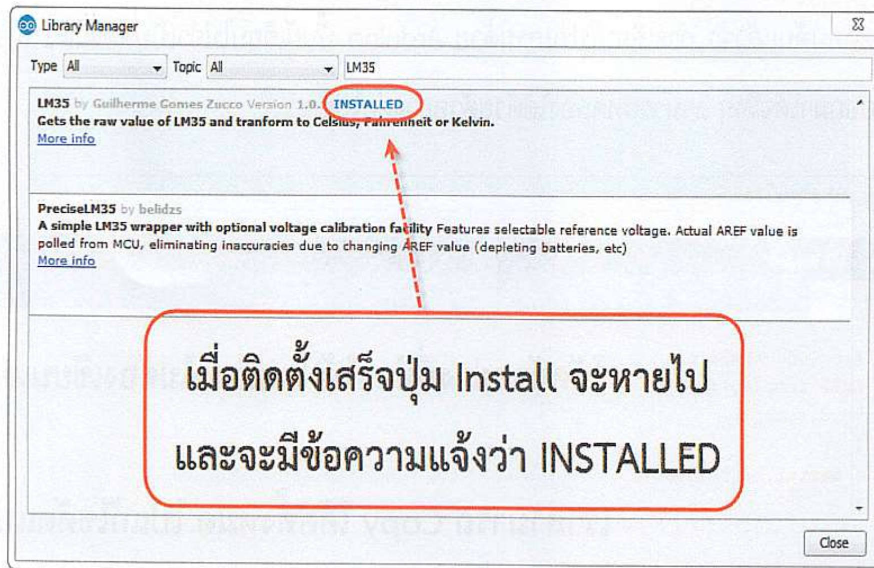
โดยมองหาเมนูชื่อ Sketch ที่เมนูบาร์ ที่อยู่ด้านบน

คลิกที่เมนู Sketch แล้วชี้เมาส์ไปที่ Include Library

เลือกคลิกที่ Manage Libraries...

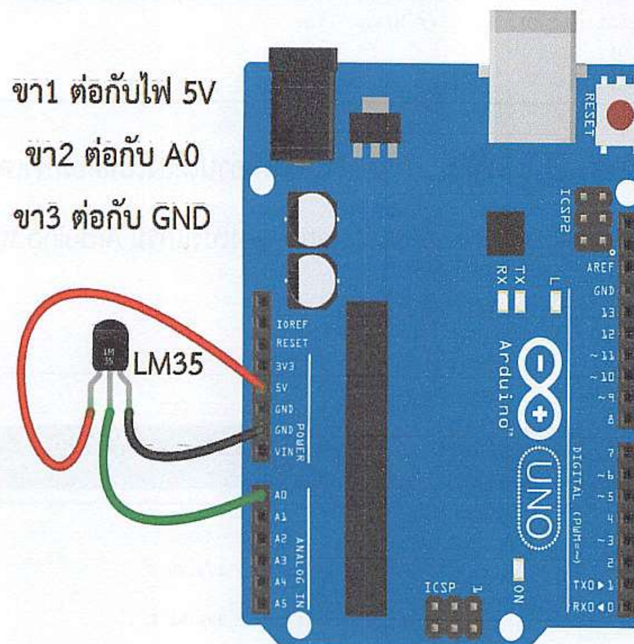






ขั้นตอนติดตั้ง Library เสร็จสมบูรณ์และพร้อมจะเรียกขึ้นมาใช้งานได้ทันที

การต่อใช้งานไอซี LM35 เข้ากับบอร์ด Arduino



การต่อวงจรวัดอุณหภูมิด้วยไอซี LM35 และส่งข้อมูลไปแสดงที่คอมพิวเตอร์

ไอซีเบอร์ LM35 เป็นไอซี Temperature Sensor เป็นเซ็นเซอร์ที่ใช้สำหรับวัดอุณหภูมิแบบแอนาล็อก จริงๆแล้วการเขียนโปรแกรมใช้งานนั้นจะต้องอ่านค่าจาก ADC แล้วเอามาคำนวณเพื่อหาค่าอุณหภูมิ แต่สำหรับการเขียนโปรแกรมด้วย Arduino นั้นมีไลบรารีสำเร็จรูปให้ใช้งานได้อย่างง่ายดายสามารถเรียกดูค่าอุณหภูมิได้ด้วยการเขียนโค้ดเพียงบรรทัดเดียวเท่านั้นเพราะการคำนวณทุกๆอย่างไลบรารีจะเป็นตัวจัดการทุกอย่าง เราเพียงแค่เรียกใช้งานไลบรารีนั้นเพื่อเรียกค่าข้อมูลออกมาแสดงผลเท่านั้นเอง และอย่างที่ผู้เขียนได้

กล่าวไว้แต่ตอนต้นแล้วว่า การเขียนโปรแกรมด้วย Arduino นั้นผู้เขียนไม่จำเป็นต้องมีความรู้มากนักก็สามารถ สร้างงานออกมาได้จริงๆ สามารถทดลองได้ด้วยตัวอย่างต่อไปนี้

Temp_lm35 | Arduino 1.8.5

File Edit Sketch Tools Help

Temp_lm35 §

```
#include <LM35.h>
LM35 temp(A0);
void setup()
{
  Serial.begin(9600);
}
void loop()
{
  Serial.print("Temp - ");  //Send a text to serial
  Serial.print(temp.cel()); //It gets the temperature in celcius and send to serial
  Serial.print(" C  - ");
  Serial.print(temp.fah()); //It gets the temperature in fahrenheit and send to serial
  Serial.print(" F  - ");
  Serial.print(temp.kel()); //It gets the temperature in kelvin and send to serial
  Serial.println(" K");     // Send a text to serial and give a new line
  Serial.println(" ");      // Blank line
  delay(1000);
}
```

โค้ดตัวอย่างที่มีมาให้ โดยที่เราไม่ต้องเขียนเองเลย

เราสามารถ Copy โค้ดทั้งหมด ไปแก้ไขดัดแปลงได้

แต่โค้ดตัวอย่างที่ได้มานั้นเป็นการส่งค่าข้อมูลที่อ่านได้ส่งไปที่คอมพิวเตอร์โดยผ่านช่องทางที่เรียกว่า Serial Port เพื่อไปแสดงผลที่ Serial Monitor ของโปรแกรม Arduino หมายความว่าต้องมีการเชื่อมต่อกับคอมพิวเตอร์เท่านั้นจึงจะทำงานได้

Temp_lm35 | Arduino 1.8.5

File Edit Sketch Tools Help

Temp_lm35 §

```
// setting the se
// Command - LM35
LM35 temp(A0);

// Function that
void setup()
{
  // start the Se
  Serial.begin(96
  pinMode(A0,INPU
}

// Function that
void loop()
{
  Serial.print("T
  Serial.print(te
  Serial.print(" C
  Serial.print(te
```

COM5

Send

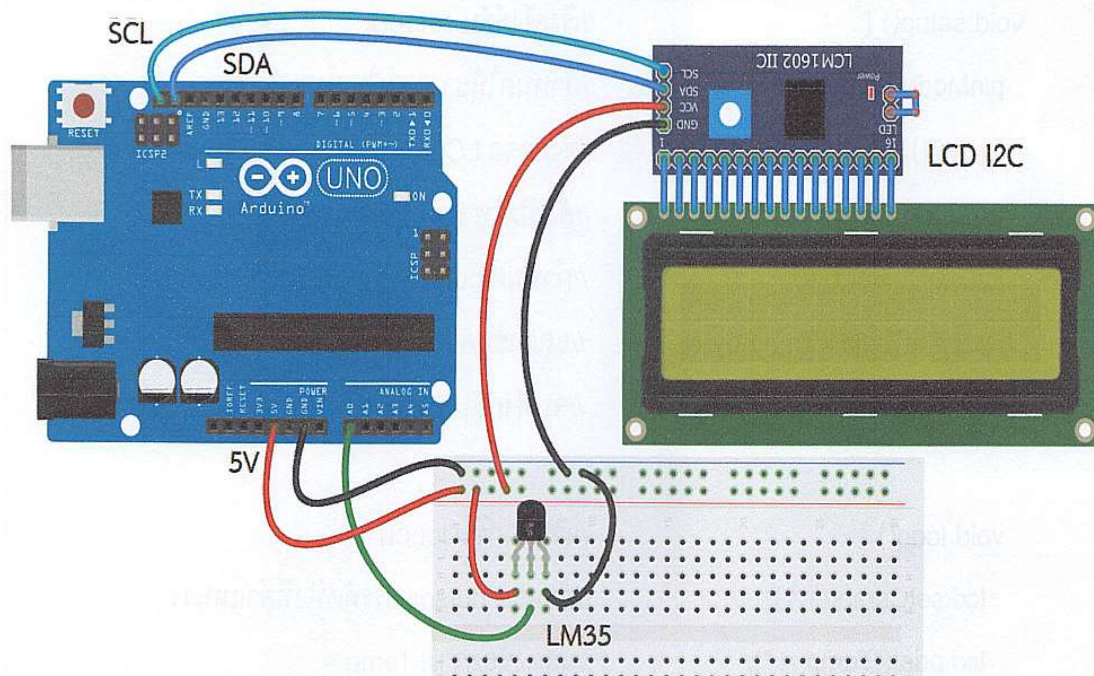
```
Temp - 29.81 C  - 86.55 F  - 303.30 K
Temp - 30.79 C  - 85.67 F  - 302.81 K
Temp - 29.81 C  - 85.67 F  - 303.30 K
Temp - 29.81 C  - 86.55 F  - 302.81 K
Temp - 30.30 C  - 86.55 F  - 302.81 K
Temp - 30.30 C  - 87.43 F  - 302.81 K
Temp - 30.30 C  - 85.67 F  - 303.30 K
```

Autoscroll　Both NL & CR　9600 baud　Clear output

ภาพแสดงตัวอย่างการอ่านค่าจาก Serial Monitor ของ Arduino

แต่ตัวอย่างต่อไปนี้จะเป็นการดัดแปลงให้แสดงผลที่จอ LCD เพื่อที่เราจะสามารถใช้งานได้โดยไม่ต้องเชื่อมต่อกับคอมพิวเตอร์อีกต่อไป โครงงานนี้จะเป็นการเชื่อมต่อกับจอ LCD แบบ I2C เพื่อเป็นการลดความยุ่งยากในการต่อสายสัญญาณระหว่างบอร์ด Arduino กับจอ LCD เพราะการต่อใช้งานแบบ I2C นั้นใช้สายสัญญาณเพียง 4 เส้นเท่านั้น แต่จะต้องดาวน์โหลดไลบรารีสำหรับใช้งานจอ LCD แบบ I2C มาติดตั้งเสียก่อนจึงจะใช้งานได้ ซึ่งขั้นตอนการดาวน์โหลดและติดตั้งไลบรารีนั้น ได้กล่าวไว้แล้วในเรื่องของ <u>การดาวน์โหลดและติดตั้งไลบรารีของเซ็นเซอร์ต่างๆ</u>

การต่อวงจรวัดอุณหภูมิด้วยไอซีอนาล็อก LM35 แสดงผลด้วย LCD16x2 แบบ I2C



//โค้ดโปรแกรมวัดอุณหภูมิด้วย LM35 แสดงผลด้วยจอ LCD แบบ I2C

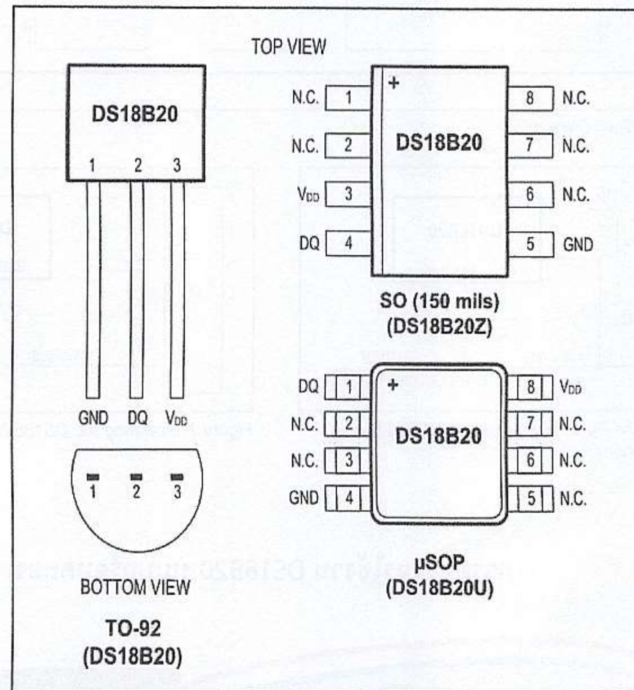
```
#include <LM35.h>                //เรียกใช้ไลบรารี LM35
#include <Wire.h>                //เรียกใช้ไลบรารี Wire
#include <LiquidCrystal_I2C.h>   //เรียกใช้ไลบรารี LCD แบบ I2C
LM35 temp(A0);                   // ใช้ฟังก์ชันสำเร็จรูปของ LM35 ให้ temp อ่านค่าที่ขา
A0
LiquidCrystal_I2C lcd(0x3F,16,2);  //กำหนดค่าแอดเดรสของจอ LCD

void setup() {                   //เริ่มฟังก์ชัน setup()
  pinMode(A0,INPUT);             //กำหนดให้ขา A0 เป็น INPUT
  lcd.init();                    //ตั้งค่าจอ LCD
  lcd.backlight();               //สั่งเปิดไฟ backlight ของจอ LCD
  lcd.setCursor(0,0);            //กำหนด cursor บรรทัดแรกที่ตำแหน่ง 0
  lcd.print("LM35 Temp by NDE");  //แสดงข้อความบรรทัดแรก
  }                              //จบฟังก์ชัน setup()

 void loop() {                   //เริ่มฟังก์ชัน loop()
   lcd.setCursor(1,1);           //กำหนด cursor บรรทัดที่2ที่ตำแหน่ง1
   lcd.print("Temp = ");         //แสดงข้อความ Temp =
   lcd.print(temp.cel());        //แสดงค่าอุณหภูมิเป็นองศา C
   lcd.print("  C ");            //แสดงตัวอักษร C
   delay(1000);                  //หน่วงเวลา 1 วินาที
   }                             //สิ้นสุดโปรแกรม
```

ตัวอย่างการใช้งานเซ็นเซอร์วัดอุณหภูมิ DS18B20

DS18B20 เป็นเซนเซอร์วัดอุณหภูมิแบบดิจิทัล ที่มาในรูปแบบของไอซี มีตัวถังให้เลือกใช้งานได้ทั้งแบบตัวไอซี 3 ขา และ 8 ขา ส่วนใหญ่จะนิยมใช้งานแบบ TO-92 มีขาใช้งาน 3 ขา เพราะการต่อใช้งานสะดวกกว่า



DC Electrical Characteristics

(-55°C to +125°C; V_{DD} = 3.0V to 5.5V)

PARAMETER	SYMBOL	CONDITIONS		MIN	TYP	MAX	UNITS
Supply Voltage	V_{DD}	Local power (Note 1)		+3.0		+5.5	V
Pullup Supply Voltage	V_{PU}	Parasite power	(Notes 1, 2)	+3.0		+5.5	V
		Local power		+3.0		V_{DD}	
Thermometer Error	t_{ERR}	-10°C to +85°C	(Note 3)			±0.5	°C
		-55°C to +125°C				±2	
Input Logic-Low	V_{IL}	(Notes 1, 4, 5)		-0.3		+0.8	V
Input Logic-High	V_{IH}	Local power	(Notes 1,6)	+2.2		The lower of 5.5 or V_{DD} + 0.3	V
		Parasite power		+3.0			
Sink Current	I_L	$V_{I/O}$ = 0.4V		4.0			mA
Standby Current	I_{DDS}	(Notes 7, 8)			750	1000	nA
Active Current	I_{DD}	V_{DD} = 5V (Note 9)			1	1.5	mA
DQ Input Current	I_{DQ}	(Note 10)			5		µA
Drift		(Note 11)			±0.2		°C

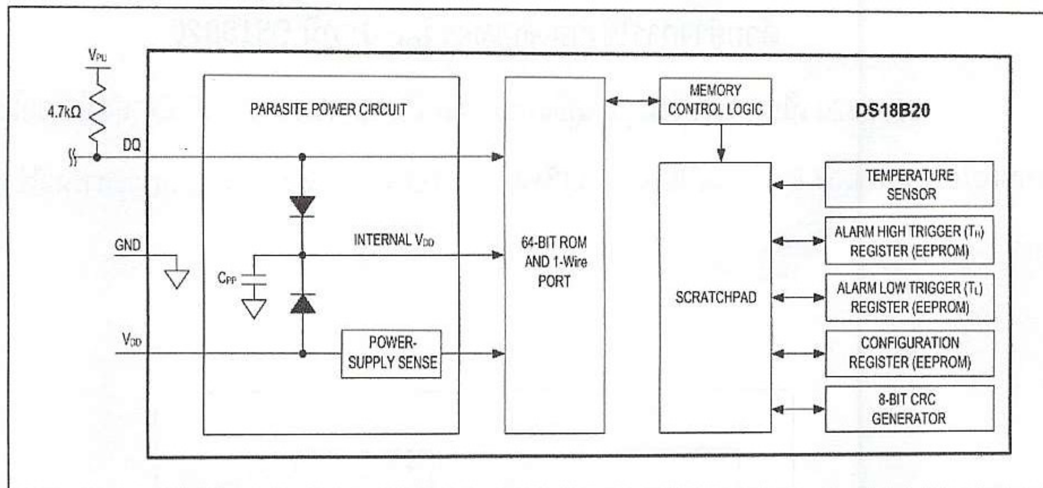


Figure 3. DS18B20 Block Diagram

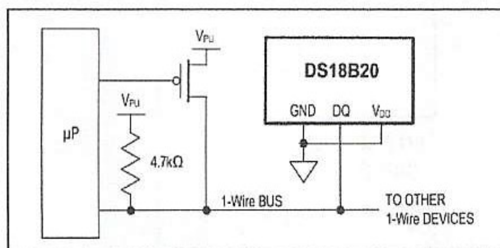


Figure 6. Supplying the Parasite-Powered DS18B20 During Temperature Conversions

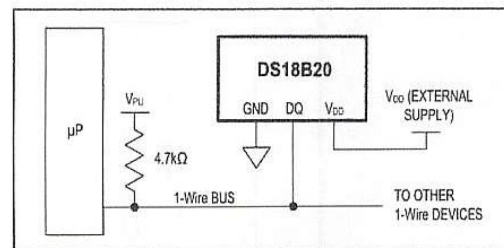
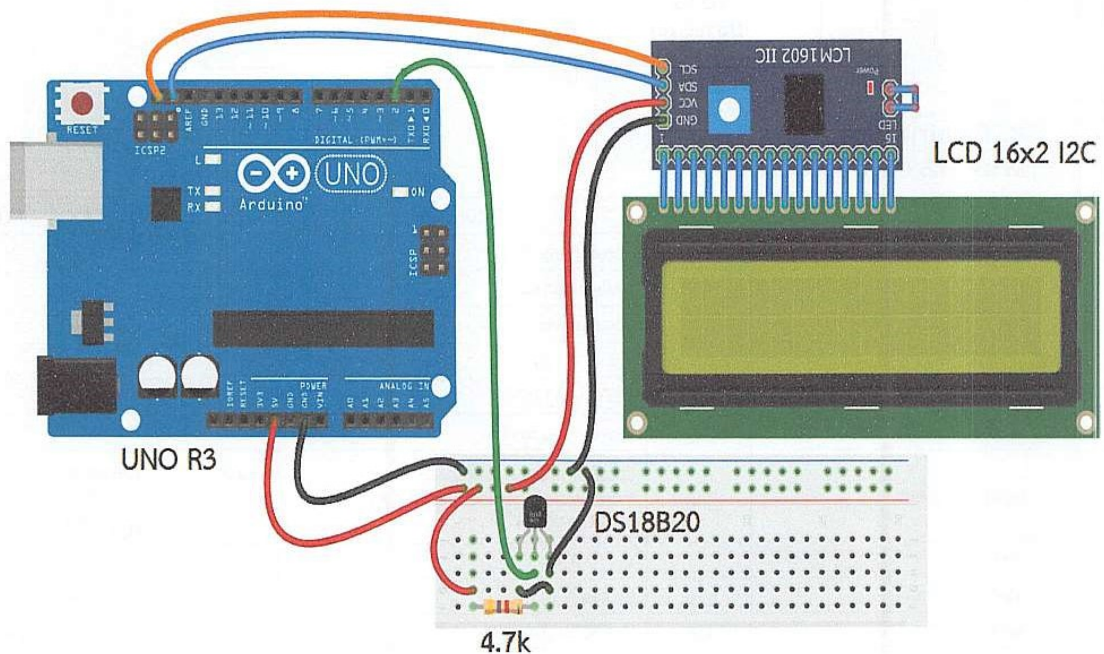


Figure 7. Powering the DS18B20 with an External Supply

การต่อวงจรใช้งาน DS18B20 บนบอร์ดทดลอง



```
//โค้ดโปรแกรมวัดอุณหภูมิด้วยเซ็นเซอร์ดิจิทัล DS18B20 แสดงผลด้วย LCD16x2 แบบ I2C
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <OneWire.h>
#include <DallasTemperature.h>
#define ds18b20 2

LiquidCrystal_I2C lcd(0x3F,16,2);
OneWire pin2(ds18b20);
DallasTemperature temp(&pin2);
float TempC;

void setup() {
  lcd.init();
  lcd.backlight();
  temp.begin();
}

void loop() {
temp.requestTemperatures();
TempC = temp.getTempCByIndex(0);
lcd.setCursor(0,0);
lcd.print("Temp =");
lcd.print(" ");
lcd.print(TempC);
lcd.print(" ");
lcd.write(0xdf);
lcd.print("C");
delay(200);
}
```

ตัวอย่างการใช้งานเซ็นเซอร์วัดอุณหภูมิด้วย เทอร์มิสเตอร์ แบบ NTC

เทอร์มิสเตอร์ (Thermistor) เป็นอุปกรณ์ประเภททรานสดิวเซอร์ อุปกรณ์ประเภททรานสดิวเซอร์นี้จะทำหน้าที่ในการแปลงค่าปรากฏการณ์ทางฟิสิกส์ เช่น อุณหภูมิ ,ความดัน ,อัตราการไหล และอื่นๆอีกมากมาย อุปกรณ์ประเภททรานสดิวเซอร์นี้จะทำการเปลี่ยนแปลงค่าต่างๆดังที่กล่าวมานั้นให้เป็นสัญญาณทางไฟฟ้าอย่างเป็นสัดส่วน และบางชนิดก็จะเปลี่ยนค่าความต้านทานไปตามค่าอุณหภูมิที่เปลี่ยนแปลง และเทอร์มิสเตอร์นั้นก็จัดอยู่ในหมวดของอุปกรณ์ทรานสดิวเซอร์ที่เปลี่ยนค่าความต้านทานไปตามอุณหภูมิ โดยการเปลี่ยนแปลงนั้นเป็นการเปลี่ยนแปลงค่าความต้านทานอย่างเป็นสัดส่วนต่ออุณหภูมิที่เปลี่ยนแปลงไป เทอร์มิสเตอร์นั้นมีอยู่ด้วยกันสองชนิด

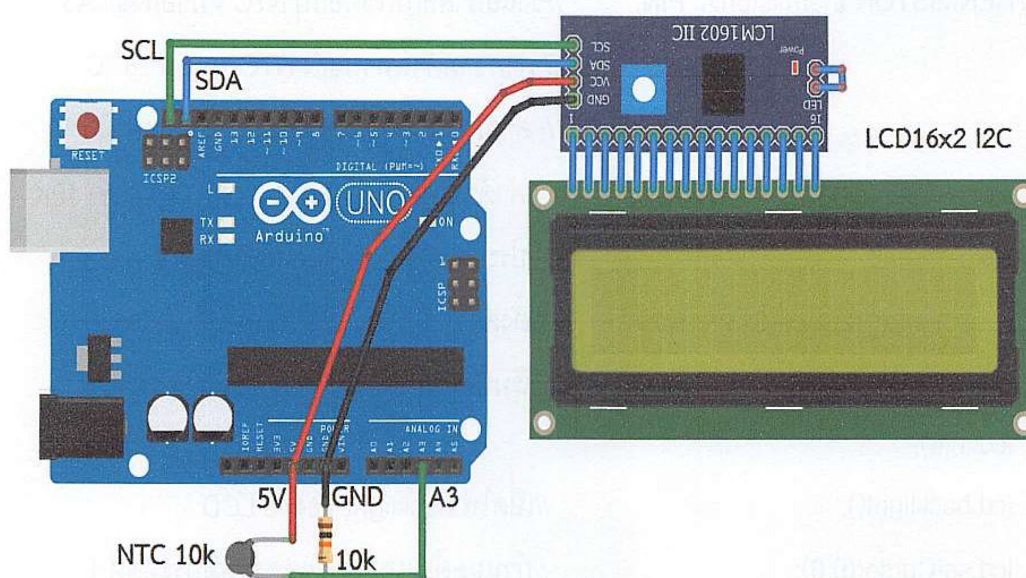
NTC (Negative Temperature Coefficient) เทอร์มิสเตอร์ชนิดนี้ถ้าหากอุณหภูมิสูงขึ้นค่าความต้านทานของมันจะลดต่ำลง โดยปกติแล้วเทอร์มิสเตอร์ชนิดนี้จะมีค่าความต้านทานที่สูงเมื่อตัวมันได้รับอุณหภูมิที่สูงขึ้น จะทำให้ค่าความต้านทานในตัวมันมีค่าลดดลง

PTC (Positive Temperature Coefficient) เทอร์มิสเตอร์ชนิดนี้ถ้าหากอุณหภูมิสูงขึ้นค่าความต้านทานของมันจะสูงขึ้นตามไปด้วย โดยปกติแล้วเทอร์มิสเตอร์ชนิดนี้จะมีค่าความต้านทานที่ต่ำ และเมื่อมันได้รับอุณหภูมิที่สูงขึ้น หมายถึงถ้าตัวมันร้อนขึ้นความต้านทานในตัวมันจะสูงขึ้นตามไปด้วย

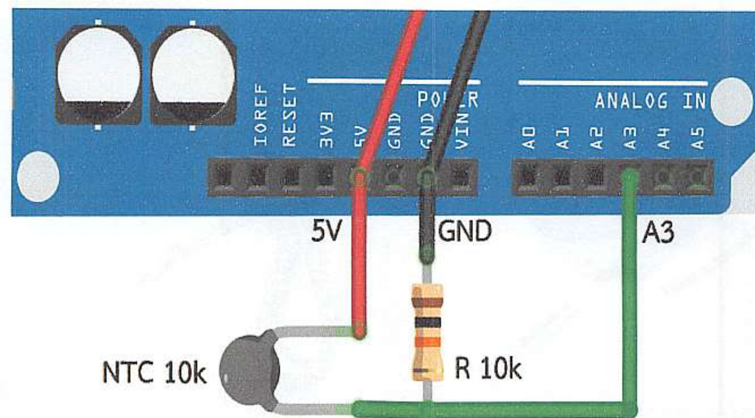
ด้วยคุณสมบัติที่เปลี่ยนแปลงค่าความต้านทานตามอุณหภูมิได้นี่เองทำให้เราสามารถนำเทอร์มิสเตอร์นั้นมาวัดอุณหภูมิได้ โดยวิธีการนั้นเป็นการนำเทอร์มิสเตอร์นั้นมาจัดเป็นวงจรแบ่งแรงดันร่วมกับรีซิสเตอร์แบบธรรมดาตัวหนึ่ง แล้วส่งค่าแรงดันไฟฟ้าที่ได้นั้นเข้าวงจร ADC ของไมโครคอนโทรลเลอร์แล้วคำนวณหาค่าความต้านทาน เพียงเท่านี้ก็จะทำให้เราทราบค่าอุณหภูมิได้อย่างง่ายดาย

ประเด็นหลักในการใช้งานเซนเซอร์หรืออุปกรณ์ต่างๆกับไมโครคอนโทรลเลอร์นั้น สิ่งสำคัญที่สุดคือเราต้องรู้จักอุปกรณ์ชนิดนั้นให้ดีเสียก่อนว่าหลักการทำงานเป็นอย่างไร ยิ่งรู้มากการที่เราจะนำไปประยุกต์ใช้งานนั้นก็จะง่ายมาก ไม่ใช่เพียงแต่เฉพาะเซนเซอร์อุณหภูมิเท่านั้น แต่รวมไปถึงอุปกรณ์ทุกชนิด การที่จะนำมาใช้งานให้เกิดประสิทธิภาพสูงสุดและง่ายดายนั้น จำเป็นต้องศึกษาทำความรู้จักอุปกรณ์นั้นให้เข้าใจหลักการทำงานก่อน แต่สิ่งที่เกิดขึ้นทุกวันนี้ เป็นการ copy ต่อๆกันมาโดยที่ไม่รู้ว่ามันคืออะไรทำงานแบบไหน รู้เพียงว่าต่อสายแบบนี้ เขียนโค้ดแบบนี้ จะให้ผลที่ต้องการ พองานเกิดปัญหาขึ้นจึงแก้ปัญหาเองไม่ได้





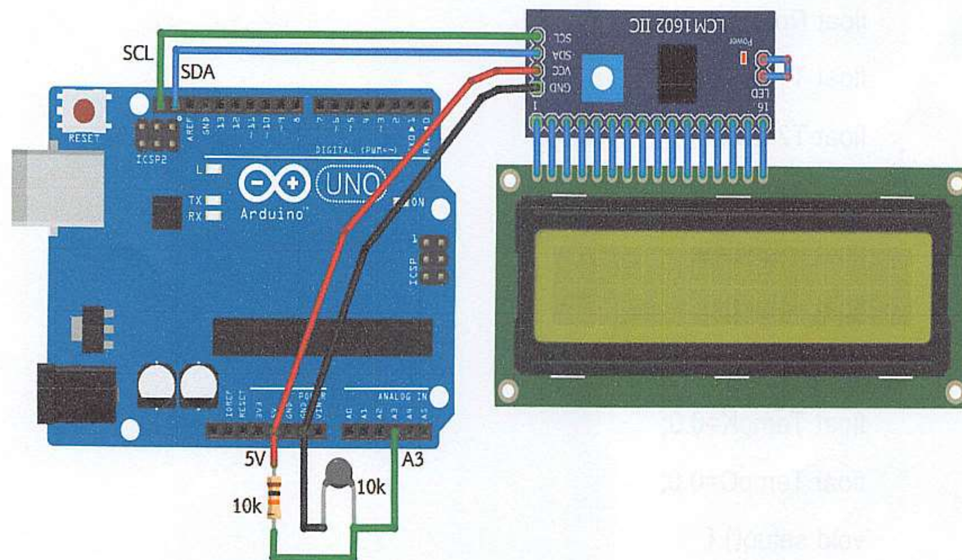
การต่อวงจรใช้งาน NTC วัดอุณภูมิ แบบใช้ Library



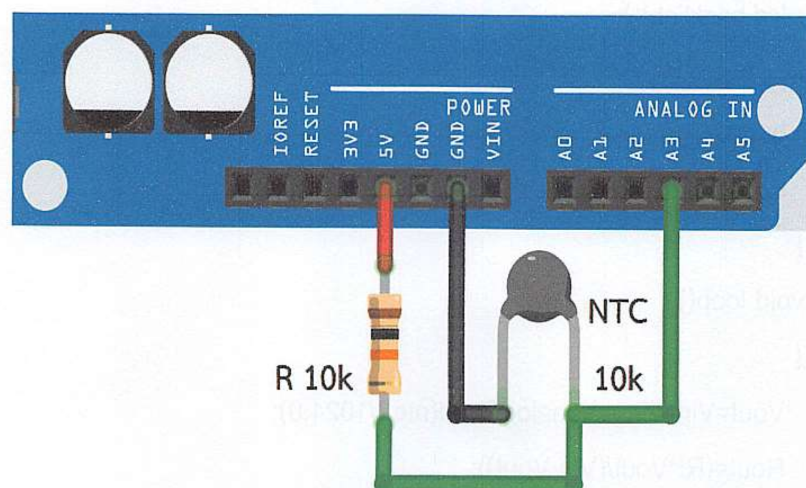
ภาพแสดงการต่อวงจรใช้งานแบบใช้ Library

```
//โค้ดโปรแกรมวัดอุณหภูมิด้วย NTC แบบใช้ Library
#include <Wire.h>                       // เรียกใช้ไลบรารี Wire เพื่อใช้งานกับ LCD แบบ I2C
#include <LiquidCrystal_I2C.h>          // เรียกใช้ไลบรารีของจอ LCD I2C
#include "thermistor.h"                 // เรียกใช้ไลบรารี thermistor
#define A_PIN A3                        // กำหนดชื่อ A_PIN ขึ้นมาใช้แทนขา A3
                                        //กำหนดค่าข้อมูลต่างๆส่งเข้าไลบรารี thermistor
THERMISTOR thermistor(A_PIN,            //ขาแอนาล็อกที่ใช้ต่อกับ NTC หมายถึงขา A3
                      10000,            // ค่าความต้านทานของ NTC 10K ที่ 25 ˚C
                      3950,             // ค่าสัมประสิทธิ์การเปลี่ยนแปลงอุณหภูมิ
                      10000);           // ค่าตัวต้านทานที่ต่อดิไวเดอร์กับ NTC ค่า 10K
uint16_t temp;                          // ประกาศตัวแปร temp เก็บค่าอุณหภูมิ
LiquidCrystal_I2C lcd(0x3F,16,2);       // ตั้งค่าแอดเดรสและการแสดงผลของจอ LCD
void setup(){                           // เริ่มฟังก์ชัน setup()
  lcd.init();
  lcd.backlight();                      //เปิดไฟ backlight ของจอ LCD
  lcd.setCursor(0,0);                   //กำหนดตำแหน่งการแสดงผลที่บรรทัดที่ 1
  lcd.print("NTC Temp by NDE");         //แสดงข้อความบรรทัดที่1
}                                       // จบฟังก์ชัน setup()
void loop(){                            //เริ่มฟังก์ชัน loop()
  temp = thermistor.read();             //อ่านค่าอุณหภูมิ ส่งเข้าในตัวแปร temp
  temp = temp/10;                       //นำข้อมูลที่อ่านได้มาหารด้วย 10
  lcd.setCursor(2,1);                   //กำหนดการแสดงผลบรรทัดที่ 2 เริ่มต้นที่ตำแหน่ง 3
```

```
lcd.print("Temp = ");                    //แสดงข้อความ Temp =
lcd.print(temp);                         //แสดงข้อมูลอุณหภูมิ
lcd.print(" ");                          //print ค่าว่างเพื่อเว้นวรรค
lcd.write(0xdf);                         //แสดงสัญลักษณ์องศา°
lcd.print("C");                          //แสดงตัวอักษร C
delay(1000);                             //หน่วงเวลา 1 วินาที
}                         //จบโปรแกรม
```



การต่อใช้งาน NTC วัดอุณหภูมิเข้ากับบอร์ด Arduino (ไม่ใช้ Library)

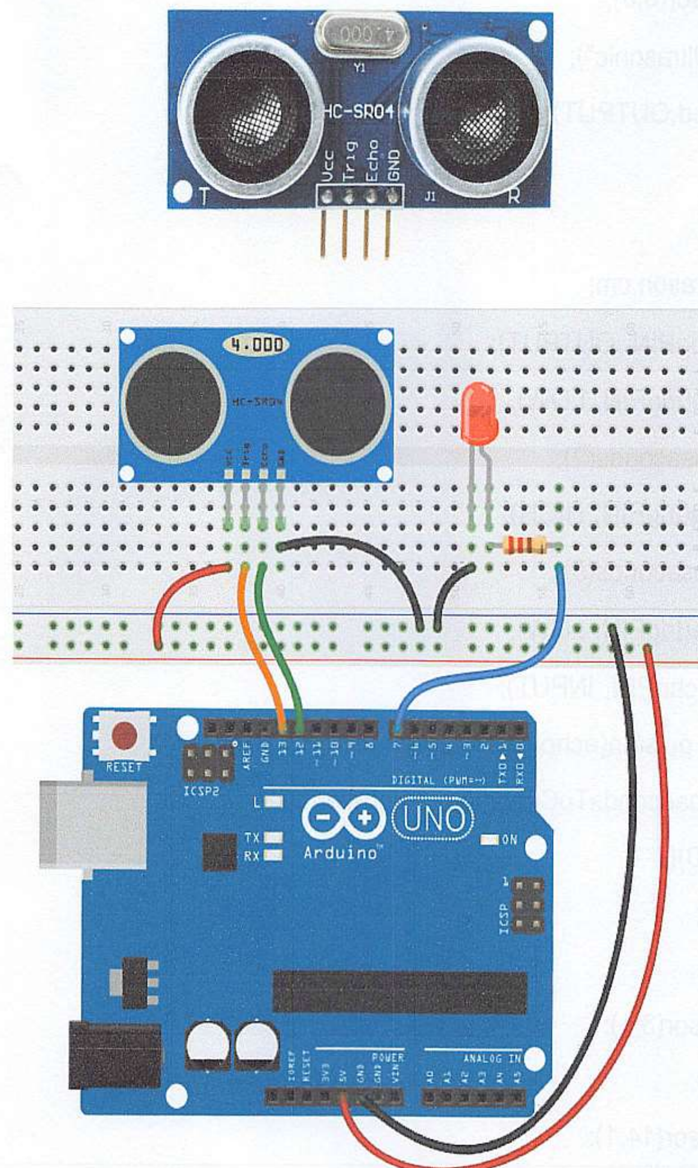


การต่อใช้งาน NTC วัดอุณหภูมิแบบไม่ใช้ Library

```
//โค้ดโปรแกรมการวัดอุณหภูมิด้วย NTC โดยไม่ใช้ไลบรารี
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#define ntc A3
LiquidCrystal_I2C lcd(0x3F,16,2);
float Vin=5.0;
float Rt=10000;
float R0=10000;
float T0=298.15;
float Vout=0.0;
float Rout=0.0;
float T1=273.15;
float T2=373.15;
float RT1=35563;
float RT2=549;
float beta=0.0;
float Rinf=0.0;
float TempK=0.0;
float TempC=0.0;
void setup() {
  lcd.init();
  lcd.backlight();
  pinMode(ntc, INPUT);
  beta=(log(RT1/RT2))/((1/T1)-(1/T2));
  Rinf=R0*exp(-beta/T0);
}
void loop()
{
  Vout=Vin*((float)(analogRead(ntc))/1024.0);
  Rout=(Rt*Vout/(Vin-Vout));
  TempK=(beta/log(Rout/Rinf));
  TempC=TempK-273.15;
  lcd.setCursor(0,0);
```

```
        lcd.print("NTC Temp by NDE");
        lcd.setCursor(0,1);
        lcd.print("Temp = ");
          lcd.print(TempC);
          lcd.print(" ");
          lcd.write(0xdf);
          lcd.print(" C");
          delay(1000);
}
```

ตัวอย่างการใช้งานเซ็นเซอร์วัดระยะทาง Ultrasonic SR-04



```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
  #define trigPIN 13
  #define led 7
  int echoPIN =12;
  float m;
  LiquidCrystal_I2C lcd(0x3F,16,2);
void setup() {
  lcd.init();
  lcd.backlight();
  lcd.setCursor(3,0);
  lcd.print("Ultrasonic");
  pinMode(led,OUTPUT);
  }
void loop() {
  double duration,cm;
  pinMode(trigPIN, OUTPUT);
  digitalWrite(trigPIN, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPIN, HIGH);
  delayMicroseconds(5);
  digitalWrite(trigPIN, LOW);
  pinMode(echoPIN, INPUT);
  duration = pulseIn(echoPIN, HIGH);
  cm = microsecondsToCentimeters(duration);
  if(cm < 100){
  int c = cm;
  lcd.clear();
  lcd.setCursor(3,1);
  lcd.print(c);
  lcd.setCursor(14,1);
```

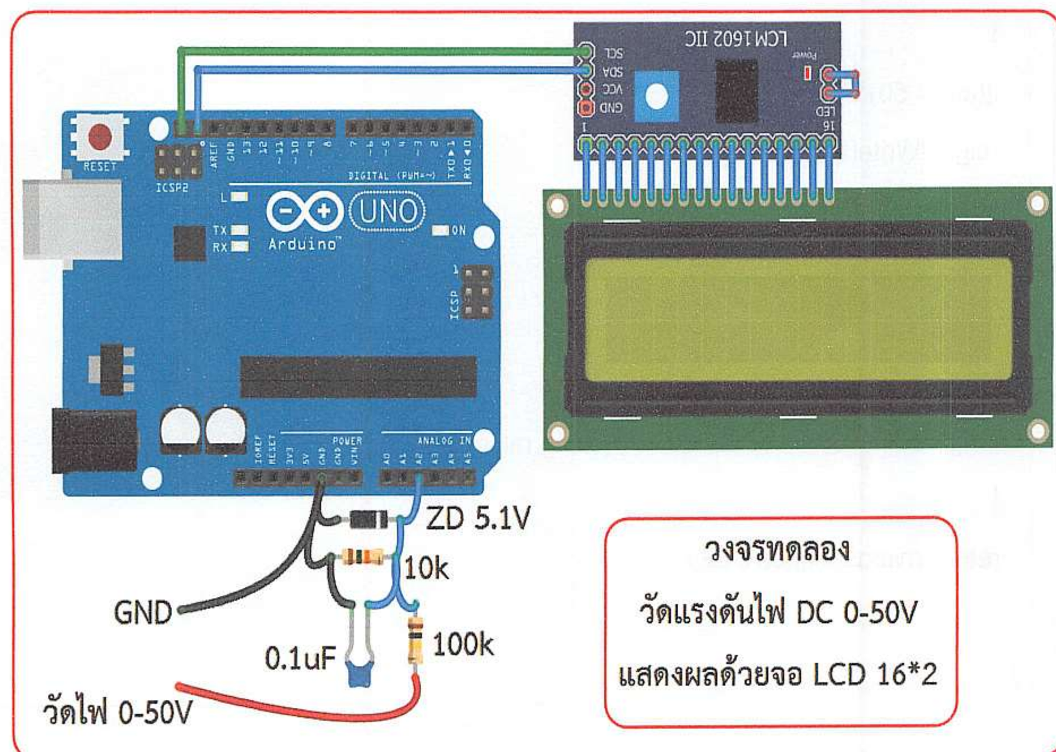
```
lcd.print("cm");
}
else if (cm >= 100){
  m = (cm/100);
lcd.clear();
lcd.setCursor(3,1);
lcd.print(m);
lcd.setCursor(14,1);
lcd.print("M");
}
if(cm < 50){
 digitalWrite(led,HIGH);
}
else digitalWrite(led,LOW);
delay(500);
}
long microsecondsToCentimeters(long microseconds)
{
return microseconds / 29 / 2;

}
```

ตัวอย่างการใช้งานวัดแรงดันไฟฟ้า DC 0-50V

การใช้งานบอร์ด Arduino เพื่อสร้างเป็นโวลต์มิเตอร์ เพื่อวัดแรงดันไฟฟ้านั้น เราจะใช้การรับค่าอินพุตเข้ามาทางขาแอนาล็อกอินพุตเพื่อนำค่าแรงดันที่รับเข้ามาเข้าสู่วงจรADC (Analog to Digital Converter) เพื่อแปลงค่าแรงดันที่เป็นสัญญาณแอนาล็อกให้เป็นสัญญาณดิจิทัลเสียก่อน จึงจะสามารถนำไปเข้าสูตรคำนวณเพื่อหาค่าแรงดันที่รับเข้ามานั้น แต่ปัญหาประการหนึ่งของการใช้ไมโครคอนโทรลเลอร์ไปวัดแรงดันไฟฟ้านั้นมีอยู่ว่า ไมโครคอนโทรลเลอร์ทั่วไปนั้นใช้ไฟเลี้ยงวงจรเพียงแค่ 1.8V ถึงไม่เกิน 5V และขาอินพุตที่ใช้รับสัญญาณเข้ามานั้นจะต้องมีแรงดันไม่เกินค่าแรงดันไฟเลี้ยงตัวไอซี

หากเราต้องการนำไมโครคอนโทรลเลอร์ไปวัดแรงดันไฟฟ้าที่เกินกว่าแรงดันไฟเลี้ยงของตัวมันเอง และบอร์ดที่เราใช้ทดลองในหนังสือเล่มนี้คือ Arduino UNO R3 ซึ่งใช้แรงดันไฟเลี้ยงวงจรที่ 5V ดังนั้นจึงต้องทำให้แรงดันที่จะวัด ที่จะรับเข้ามาทางขาอินพุตนั้นมีแรงดันไม่เกิน 5V ด้วยวงจรแบ่งแรงดันธรรมดานี่เอง จากรูปวงจรตัวอย่างจะเห็นได้ว่าไม่ได้มีแค่ R เพื่อแบ่งแรงดันเพียงสองตัวเท่านั้น ยังมีตัวเก็บประจุ ค่า 0.1uF ต่อไว้เพื่อให้แรงดันที่วัดนั้นมีค่าที่นิ่ง และยังมีซีเนอร์ไดโอดต่อป้องกันไว้ที่ขาอินพุตเพื่อป้องกันความเสียหายเอาไว้อีกชั้นหนึ่ง หากนำไปวัดแรงดันที่เกินกว่า 50 โวลต์ แรงดันที่เข้ามาทางขาอินพุตจะถูกรักษาไว้ที่ไม่เกิน 5.1 โวลต์ด้วยซีเนอร์ไดโอด เป็นการเพิ่มความปลอดภัยให้ตัวไอซีไมโครคอนโทรลเลอร์



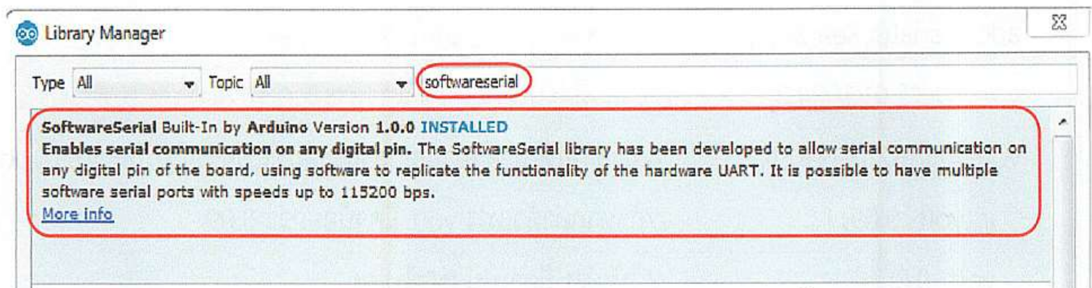

```
#include <Wire.h>                    //เรียกใช้ไลบรารี Wire
#include <LiquidCrystal_I2C.h>       //เรียกใช้ไลบรารี LCD I2C
LiquidCrystal_I2C lcd(0x3F,16,2);    //กำหนด Address และ ขนาดของจอ LCD
float v = 0.0;                       //ประกาศตัวแปร v เพื่อใช้เก็บค่าที่อ่านได้จาก ADC
float volt = 0.0;                //ประกาศตัวแปร volt เพื่อใช้เก็บค่าแรงดันที่ได้จากการคำนวณ
float R1 = 100000.0;                 //ประกาศตัวแปรเก็บค่า R1 ค่า 100k = 100000 โอห์ม
float R2 = 10000.0;                  //ประกาศตัวแปรเก็บค่า R2 ค่า 10k = 10000 โอห์ม
int adc = 0;                         //ประกาศตัวแปร adc เพื่อเก็บค่าที่อ่านได้จาก anlogRead()
void setup(){                        //เริ่มฟังก์ชัน setup
```

```
  lcd.init();                      //เริ่มต้นการแสดงผลด้วย LCD
  lcd.backlight();                 //เปิดไฟพื้นหลังของขอ LCD
  pinMode(A2, INPUT);              //กำหนดให้ขา A2 ทำงานเป็นอินพุต
}
void loop(){                       //เริ่มฟังก์ชัน loop
  adc = analogRead(A2);            //อ่านค่าจาก ADC เก็บในตัวแปร value
  v = (adc*5.0)/1024.0;            //นำค่าที่อ่านได้คูณ 5.0 แล้วหารด้วย 1024.0 เก็บในตัวแปร v
  volt = v/(R2/(R1+R2));           //นำค่าที่ได้ มาคำนวณหาค่าแรงดันที่แท้จริงเก็บในตัวแปร volt
    if (volt<0.09) {               //ถ้าหากค่าตัวแปร volt มีค่าน้อยกว่า 0.09
    volt=0.0;                      // ให้volt มีค่าเป็น 0.0 ไปเลย
    }
  lcd.setCursor(1,0);              //กำหนดให้เริ่มแสดงข้อความที่บรรทัดแรก ตำแหน่งที่ 2
  lcd.print("NDE Volt Meter");     //แสดงข้อความ  "NDE Volt Meter" ที่บรรทัดแรก
  lcd.setCursor(0,1);              //กำหนดให้เริ่มแสดงข้อความที่บรรทัดสอง ตำแหน่งแรก
  lcd.print("Voltage = ");         // แสดงข้อความ "Voltage = "
  lcd.print(volt);                 //แสดงค่าแรงดันจริงที่วัดได้
  lcd.print("V");                  // ตามหลังด้วยการแสดงตัวอักษร "V"
  delay(500);                      //หน่วงเวลา 500 มิลลิวินาที
  lcd.clear();                     //เคลียหน้าจอ เพื่อรอรับค่าใหม่
}                                  // สิ้นสุดโปรแกรม
```

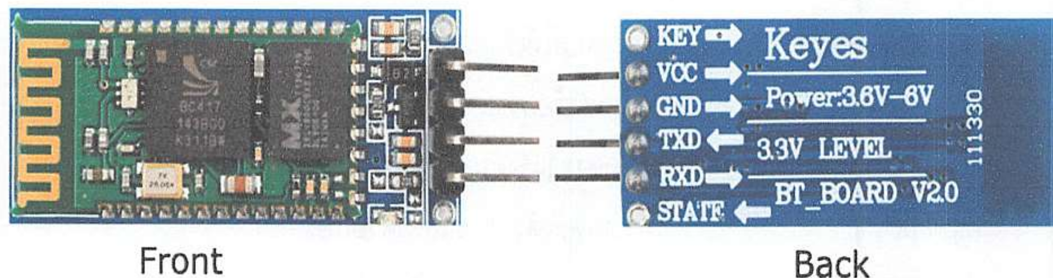
รีโมท 4 ช่องสั่งงานด้วยสมาร์ทโฟน ผ่าน Bluetooth HC-06

โครงงานรีโมท 4 ช่อง สั่งงานด้วยโทรศัพท์มือถือ Android โดยการส่งข้อมูลผ่าน Bluetooth ด้วยแอพพลิเคชั่นที่เราเขียนขึ้นมาเอง สิ่งที่จะใช้ในการรับ-ส่งข้อมูลระหว่างโทรศัพท์กับบอร์ด Arduino นั้นจะต้องใช้โมดูล Bluetooth สำเร็จรูปเข้ามาช่วย เพราะว่าลำพังบอร์ด Arduino เองนั้นไม่มีระบบการรับ-ส่งข้อมูลแบบไร้สาย ในส่วนของการรับ-ส่งข้อมูลระหว่างโมดูล Bluetooth กับบอร์ด Arduino นั้นจะใช้การรับส่งข้อมูลผ่าน Serial Port แบบ USART ใช้สายในการรับส่งข้อมูลเพียงสองเส้น คือ Tx และ Rx แต่เนื่องจากว่าบอร์ดรุ่น UNO R3 มี USART มาให้เพียงช่องเดียวและได้มีการใช้งานไปแล้วสำหรับใช้ในการเชื่อมต่อกับ USB ของคอมพิวเตอร์

เพื่อใช้ในการอัพโหลดโปรแกรม แต่เรายังสามารถใช้งานได้ด้วยการเขียนโปรแกรมเชื่อมต่อเอาต่างหากโดยใช้ขา I/O ธรรมดา เรียกว่า Software Serial แต่ในที่นี้เราไม่จำเป็นต้องเขียนขึ้นมาเอง เพราะมีไลบรารีสำเร็จรูปที่ Arduino มีให้เราดาวน์โหลดมาใช้งานได้ทันที ชื่อว่า SoftwareSerial สามารถเข้าไปดาวน์โหลดได้จาก Library Manager

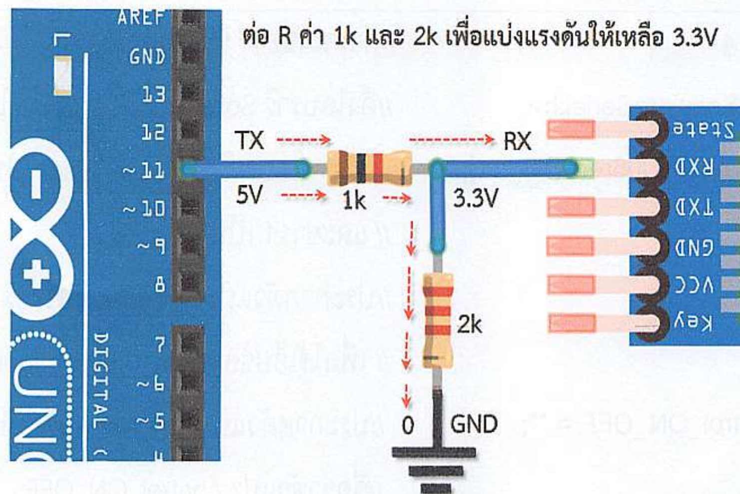


ทำการดาวน์โหลดไลบรารี SoftwareSerial มาติดตั้งก่อน



หน้าตาโมดูล Bluetooth HC-06

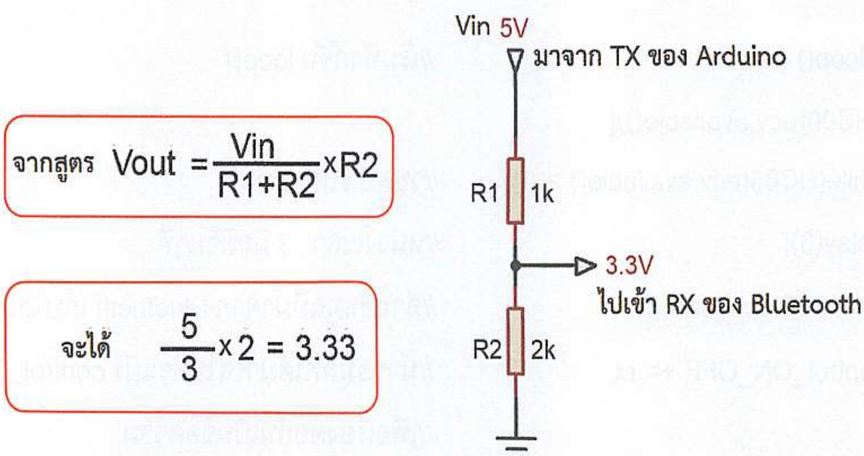
ในการต่อบอร์ด Arduino เข้ากับโมดูล Bluetooth นั้นถึงแม้ว่าขา VCC ของโมดูล Bluetooth จะใช้ต่อกับไฟเลี้ยง VCC 5Vจากบอร์ด Arduino แต่ไฟเลี้ยงตัวชิพจริงๆนั้นอยู่ที่ 3.3V เท่านั้น และมีการเขียนไว้อย่างชัดเจนบนตัวบอร์ดว่า LEVEL: 3.3V หมายถึงระดับสัญญาณที่ใช้รับส่งข้อมูลนั้นไม่เกิน 3.3V แต่ OUTPUT ของบอร์ด Arduino UNO R3 นั้นมีแรงดันไฟ 5V หากส่งสัญญาณนี้ไปเข้าขา RX ของโมดูล Bluetooth ย่อมไม่เป็นผลดีต่อโมดูลอย่างแน่นอน ดังนั้นเราจึงต้องมีการต่อวงจร Divider เพื่อแบ่งแรงดัน ให้ระดับสัญญาณนั้นมีแรงดันไม่เกิน 3.3V โดยการใช้ R ค่า 1k และ 2k มาต่อให้เป็นวงจรแบ่งแรงดันให้เหลือ 3.3V



5V ในที่นี้เป็นการกล่าวถึงระดับแรงดันไฟที่ออกมาจากขาเอาต์พุต Tx ไม่ใช่การต่อไฟ 5V เข้ามาที่ขานี้

หากบอร์ดที่ใช้เป็นบอร์ดที่ใช้ไฟเลี้ยง 3.3V ก็ไม่มีความจำเป็นต้องต่อ R แบ่งแรงดันดังกล่าว

ภาพตัวอย่างการต่อ R แบ่งแรงดัน สัญญาณ TX ที่มีแรงดัน 5V จากบอร์ด Arduino จะไหลผ่าน R 1k ก่อนที่จะเข้าไปที่ ขา RX ของโมดูล Bluetooth และกระแสส่วนหนึ่งจะไหลผ่าน R 2k ต่อลงกราวด์ไป ส่งผลให้แรงดันที่ขา RX ของโมดูล Bluetooth นั้นจะเหลือแรงดันอยู่ที่ประมาณ 3.3V ตามสูตรของการทำงานของวงจร Divider หรือ วงจรแบ่งแรงดันนั่นเอง



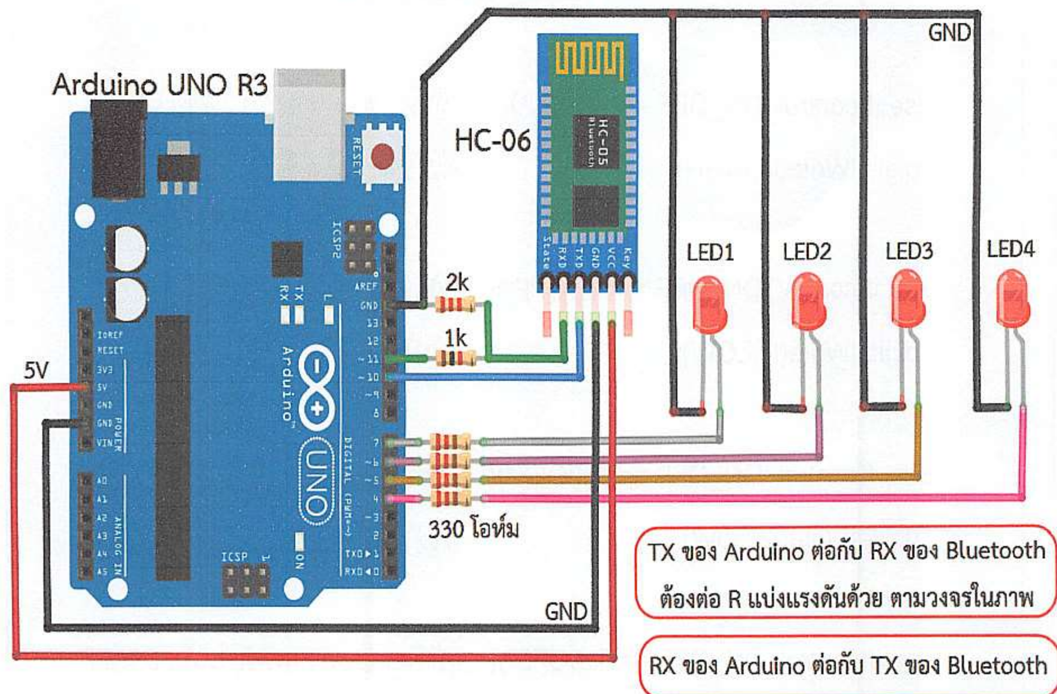
//โค้ดโปรแกรม ฝั่ง Arduino	
#define I1 7	//กำหนดชื่อ I1 ขึ้นมาแทนขา 7
#define I2 6	//กำหนดชื่อ I2 ขึ้นมาแทนขา 6
#define I3 5	//กำหนดชื่อ I3 ขึ้นมาแทนขา 5

```
#define l4 4                              //กำหนดชื่อ l4 ขึ้นมาแทนขา 4
#include <SoftwareSerial.h>               //ดึงไลบรารี SoftwareSerial เข้ามาใช้งาน
SoftwareSerial HC06recv(10, 11);          //ใช้งานไลบรารี SoftwareSerial ให้ขา 10 เป็น RX
                                          // และขา11 เป็น TX
char rx;                                  //ประกาศตัวแปรชนิด char ชื่อว่า rx
                                          // เพื่อใช้เก็บข้อมูลที่รับมาจาก Bluetooth
String control_ON_OFF = "";               //ประกาศตัวแปรชนิด Srting เพื่อเก็บข้อความ
                                          //ชื่อว่าตัวแปร control_ON_OFF
void setup() {                            //เริ่มฟังก์ชัน setup()
  pinMode(l1,OUTPUT);                     //กำหนดให้ขา 7 ทำงานเป็นเอาต์พุต
  pinMode(l2,OUTPUT);                     //กำหนดให้ขา 6 ทำงานเป็นเอาต์พุต
  pinMode(l3,OUTPUT);                     //กำหนดให้ขา 5 ทำงานเป็นเอาต์พุต
  pinMode(l4,OUTPUT);                     //กำหนดให้ขา 4 ทำงานเป็นเอาต์พุต
  HC06recv.begin(9600);                   //กำหนดความเร็วในการรับส่งข้อมูล Serial เป็น 9600
}                                         //จบฟังก์ชัน setup()

void loop() {                             //เริ่มฟังก์ชัน loop()
  if (HC06recv.available()){
    while(HC06recv.available() > 0){      //วนลูป รับข้อมูล
    delay(3);                             //หน่วงเวลา  3 มิลลิวินาที
    rx=HC06recv.read();                   //อ่านข้อมูลที่มาจาก Bluetooth เก็บใส่ตัวแปร rx
    control_ON_OFF += rx;                 //นำข้อมูลที่ได้มาใส่ในตัวแปร control_ON_OFF
                                          //เพื่อเรียงต่อกันเป็นข้อความ
     if(control_ON_OFF == "L1ON"){        //ถ้าหากข้อความที่รับได้คือ L1ON
        digitalWrite(l1,HIGH);                 //ให้ไฟดวงที่ 1 ติด
     }
     else if(control_ON_OFF == "L2ON"){   //ถ้าหากข้อความที่รับได้คือ L2ON
        digitalWrite(l2,HIGH);                 //ให้ไฟดวงที่ 2 ติด
     }
     else if(control_ON_OFF == "L3ON"){   //ถ้าหากข้อความที่รับได้คือ L3ON
```

```
      digitalWrite(l3,HIGH);                      //ให้ไฟดวงที่ 3 ติด
    }
    else if(control_ON_OFF == "L4ON"){      //ถ้าหากข้อความที่รับได้คือ L4ON
      digitalWrite(l4,HIGH);                      //ให้ไฟดวงที่ 4 ติด
    }
    else if(control_ON_OFF == "L1OFF"){    //ถ้าหากข้อความที่รับได้คือ L1OFF
      digitalWrite(l1,LOW);                       //ให้ไฟดวงที่ 1 ดับ
    }
    else if(control_ON_OFF == "L2OFF"){    //ถ้าหากข้อความที่รับได้คือ L2OFF
      digitalWrite(l2,LOW);                       //ให้ไฟดวงที่ 2 ดับ
    }
    else if(control_ON_OFF == "L3OFF"){    //ถ้าหากข้อความที่รับได้คือ L3OFF
      digitalWrite(l3,LOW);                       //ให้ไฟดวงที่ 3 ดับ
    }
    else if(control_ON_OFF == "L4OFF"){    //ถ้าหากข้อความที่รับได้คือ L4OFF
      digitalWrite(l4,LOW);                       //ให้ไฟดวงที่ 4 ดับ
    }
  }
   control_ON_OFF="";                         //เคลียร์ค่าตัวแปรให้ว่างรอรับข้อมูลใหม่
 }
}                                   //สิ้นสุดโปรแกรม
```

การต่อวงจรเพื่อใช้ในการทดลอง





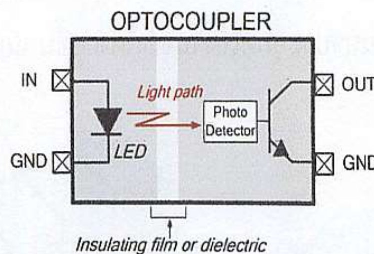
หน้าตาแอพพลิเคชั่นที่ใช้ในการทดลอง

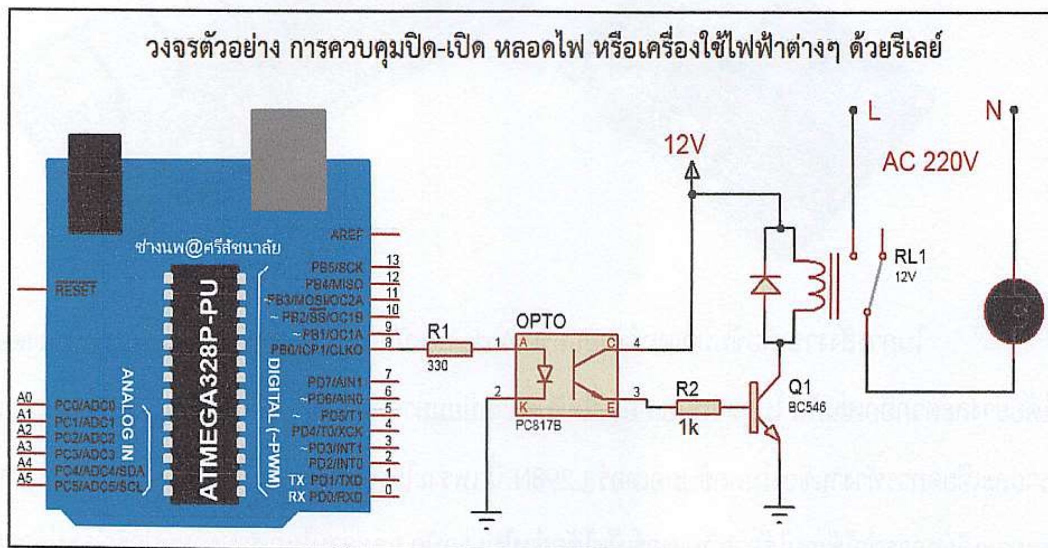


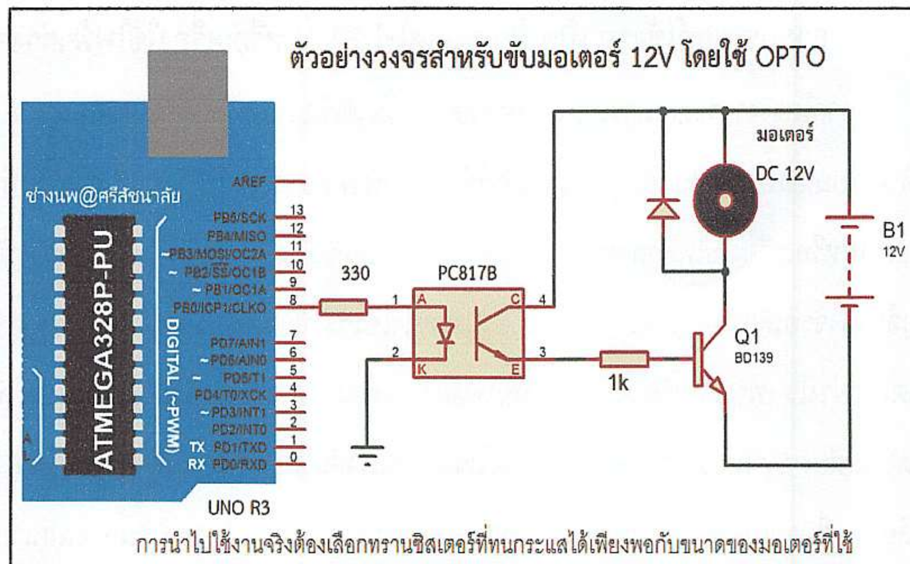
QR Code สำหรับดาวน์โหลดแอพพลิเคชั่น

การประยุกต์ใช้งาน เปิด-ปิด หลอดไฟ 220V หรือเครื่องใช้ไฟฟ้าต่างๆ

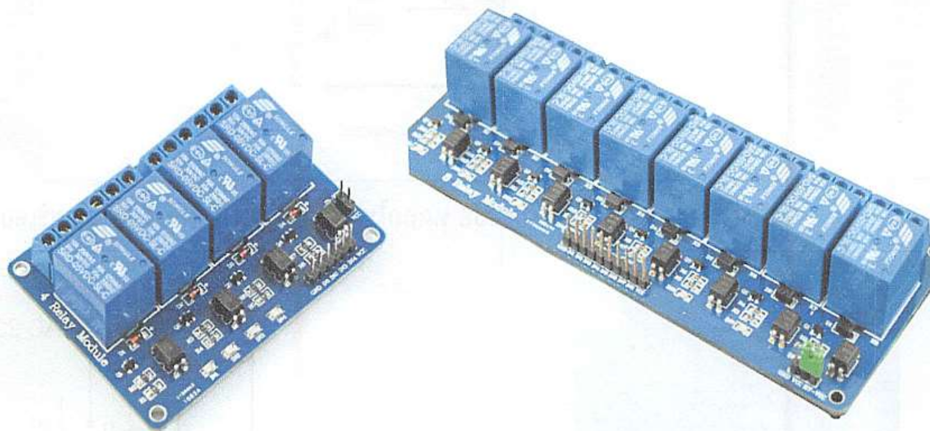
จากโครงงานที่ผ่านมาจะเห็นว่าการทดลองนั้นใช้เพียงหลอด LED ในการทดลอง และปัญหาส่วนใหญ่ที่ตัวผู้เขียนเองได้พบเห็น คือ ผู้ที่เริ่มต้นหัดใช้งานนั้นพอคิดอยากจะนำไปต่อใช้งานจริงเพื่อขับโหลดจริงๆ เช่นหลอดไฟ หรือเครื่องใช้ไฟฟ้าต่างๆ รวมไปถึงการขับมอเตอร์เพื่อนำไปใช้ประโยชน์ได้จริงๆนั้น สำหรับผู้ที่ไม่ชำนาญด้านวงจรอิเล็กทรอนิกส์มากนักจะไม่สามารถนำไปใช้งานจริงได้ เพราะไม่รู้จะเอาบอร์ด Arduino ไปขับโหลดได้อย่างไร เพราะบอร์ด Arduino นั้นใช้ไฟเลี้ยงเพียง 5V เท่านั้น ดังนั้นแรงดันไฟฟ้าที่ออกมาจากขาเอาต์พุตจึงมีเพียง 5V เท่านั้น การที่จะเอาไปขับโหลดที่ใช้แรงดันไฟสูงกว่าไฟเลี้ยงบอร์ด จึงต้องมีวงจรป้องการความเสียหายที่อาจเกิดขึ้นได้ เพราะการนำเอาต์พุตของบอร์ดไปขับโหลดที่กินกระแสมาก หรือต่อกับไฟแรงดันสูงจะทำให้ไอซีไมโครคอนโทรลเลอร์นั้นได้รับความเสียหายได้ ซึ่งวิธีการนั้นก็มีหลายวิธี แต่วิธีหนึ่งที่นิยมใช้กันมาก คือการแยกวงจรขับโหลดกับวงจรของ Arduino ออกจากกัน โดยการใช้อุปกรณ์ที่เป็นไอโซเลเตอร์ คือทั้งสองวงจรจะไม่มีการสัมผัสกัน โดยการส่งสัญญาณควบคุมผ่านแสงด้วยอุปกรณ์ที่เรียกว่า ออปโต้คลัปเปลอร์ (Optocloupler)



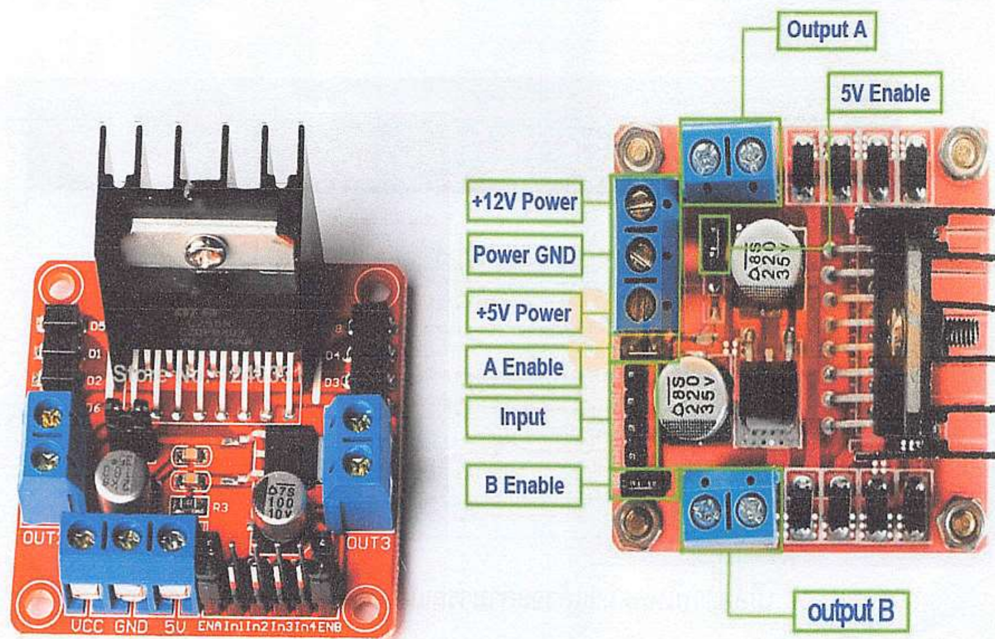




นี่เป็นตัวอย่างวงจรใช้งานอย่างง่ายๆบางส่วนเท่านั้น แต่ปัจจุบันนี้ผู้ใช้งานไม่จำเป็นต้องต่อวงจรใช้งานเองก็ได้เนื่องจากมีผู้ออกแบบวงจรสำเร็จรูปออกมาขายในราคาถูกมากผู้ใช้งานสามารถซื้อวงจรสำเร็จรูปมาต่อใช้งานได้อย่างสะดวกง่ายดาย ถึงแม้จะมีวงจรสำเร็จรูปให้ใช้งานได้อย่างสะดวกสบาย แต่การเรียนรู้การทำงานของวงจร ถือเป็นเรื่องจำเป็นที่ผู้ใช้งานจะต้องศึกษาเรียนรู้ให้เข้าใจในหลักการทำงานของวงจรอย่างแท้จริง จึงจะทำให้การนำวงจรนั้นมาประยุกต์ใช้งานจริงได้ง่ายขึ้น และสามารถใช้งานได้หลากหลายรูปแบบ



ในการสั่งงานเพื่อขับมอเตอร์ด้วยบอร์ด Arduino นั้นก็มีโมดูลสำเร็จรูปให้สามารถนำมาต่อใช้งานได้อย่างสะดวกอีกเช่นกัน โมดูลขับมอเตอร์ที่ได้รับความนิยมมากคือ L298N ในหนังสือเล่มนี้จะยังไม่กล่าวถึงรายละเอียดการทำงานของโมดูลขับมอเตอร์ L298N นี้ เพราะโมดูลสำเร็จรูปนั้นผู้ใช้งานสามารถค้นหาข้อมูลรายละเอียดการต่อใช้งานได้จากอินเตอร์เน็ตได้อย่างไม่ยากนัก และส่วนใหญ่ร้านขายอุปกรณ์จะมีตัวอย่างการต่อใช้งานมาให้อย่างละเอียด หรืออาจจะแถมโค้ดโปรแกรมตัวอย่างมาให้อีกด้วย และอีกประการหนึ่งคือข้อมูลตามเว็บไซต์ต่างๆจะเป็นภาพสีอย่างชัดเจนมากกว่าหนังสือ จะทำให้ผู้ใช้งานมองเห็นได้ชัดกว่า

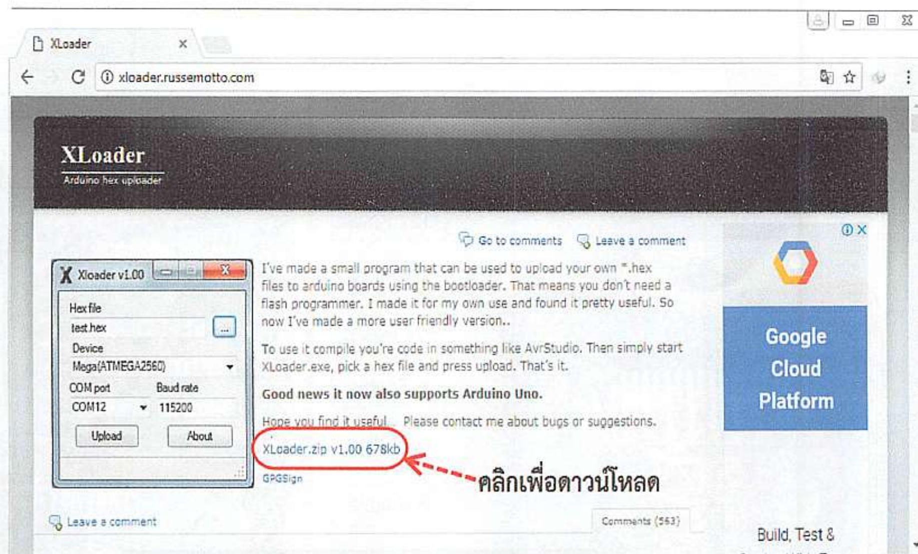


โปรแกรม Xloader สำหรับอัพโหลดไฟล์ hex ลงบอร์ด Arduino

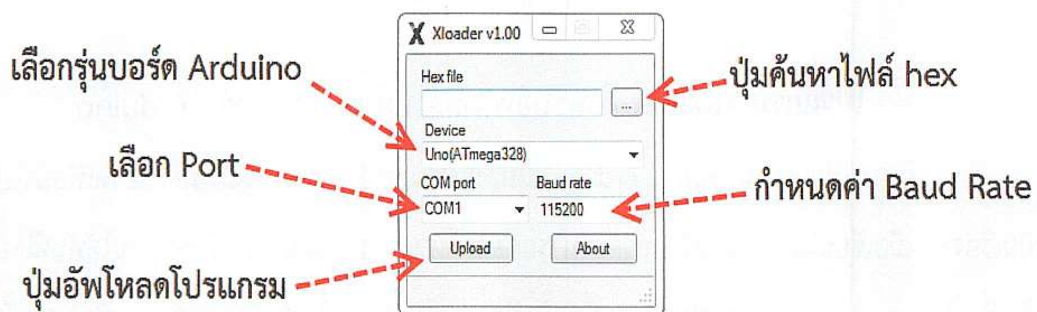
ในการเขียนโปรแกรมด้วย Arduino นั้นถึงแม้ว่าการอัพโหลดโปรแกรมลงบอร์ดเพื่อนำไปใช้งานจะเป็นเรื่องง่ายเมื่อเขียนโปรแกรมเสร็จแล้วก็สามารถกดอัพโหลดโปรแกรมลงบอร์ดให้ทำงานได้ทันทีโดยไม่ต้องใช้เครื่องโปรแกรม และไม่ต้องใช้โปรแกรมอื่นเพื่ออัพโหลดไฟล์ที่คอมไพล์แล้วลงบอร์ด แต่ในบางครั้งเราอาจมีความจำเป็นบางอย่างที่ต้องใช้วิธีการเบิร์นไฟล์ hex ลงบอร์ด Arduino เช่นในกรณีที่มีคนมาขอโปรแกรมของเราเพื่อนำไปใช้เราก็อาจจะคอมไพล์เป็นไฟล์ hex แล้วส่งให้เฉพาะไฟล์ hex โดยที่เราไม่ต้องให้โค้ดโปรแกรมที่เราเขียนขึ้นมาก็ได้ หรือในทางกลับกันเราอาจจะไปขอโปรแกรมคนอื่นมาแล้วเขาส่งมาให้เราเฉพาะไฟล์ hex หรืออาจจะใช้ในการอัพโหลดโปรแกรมที่เขียนขึ้นมานั้นลงในบอร์ดต่างรุ่นกันแต่ใช้ชิพเบอร์เดียวกัน เราสามารถที่จะคอมไพล์เป็นไฟล์ hex แล้วอัพโหลดเพื่อใช้งานกับบอร์ดอีกรุ่นหนึ่งก็ได้ เช่น บอร์ด UNO R3 กับ NANO V3 ที่ใช้ชิพ Amega328 เหมือนกัน แต่ว่าโปรแกรม Arduino IDE นั้นไม่สามารถที่จะนำเข้าไฟล์ hex จากภายนอกมาอัพโหลดลงบอร์ดได้ เราจึงต้องหาตัวช่วยในการทำงานนี้ และโปรแกรม Xloader คือโปรแกรมที่จะมาตอบโจทย์นี้ด้วยการใช้งานที่แสนง่าย

โปรแกรม Xloader เป็นโปรแกรมที่มีไว้สำหรับอัพโหลดไฟล์ hex ลงบอร์ด Arduino โดยเฉพาะ เป็นโปรแกรมฟรีแวร์สามารถหาดาวน์โหลดมาใช้ได้ฟรีๆที่เว็บไซต์ของผู้พัฒนา

http://xloader.russemotto.com/



เมื่อดาวน์โหลดมาแล้วจะสามารถแตกไฟล์แล้วเปิดใช้งานได้ทันที



โปรแกรม Xloader ในเวอร์ชั่น 1.00 นี้สามารถใช้งานได้กับบอร์ด Arduino ได้ 5 รุ่น

Mega(ATMEGA1280)

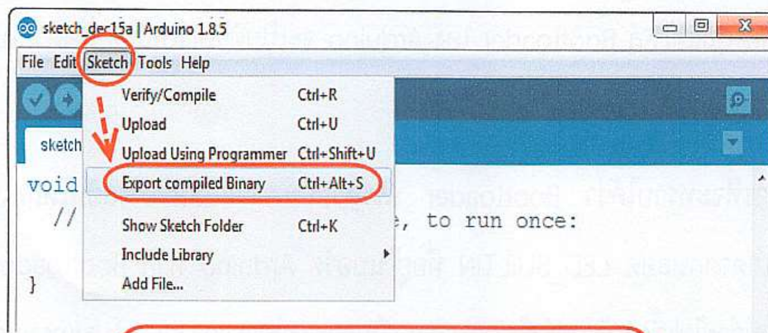
Duemilanove/Nano(ATmega328)

Duemilanove/Nano(ATmega168)

Uno(ATmega328)

Mega(ATMEGA2560)

การสร้างไฟล์ hex นั้น เมื่อเขียนโปรแกรมเสร็จแล้วจะต้องสร้างโฟลเดอร์เพื่อ save โปรแกรมไว้เสียก่อนเพราะว่าเมื่อทำการคอมไพล์แล้ว ไฟล์ hex ที่คอมไพล์ออกมานั้นจะอยู่ในโฟลเดอร์เดียวกัน เวลาจะนำมาใช้จะได้หาได้สะดวกและรวดเร็ว การสร้างไฟล์ hex ทำได้โดยการคลิกที่เมนู Sketch แล้วเลือกที่ Export compiled Binary ถ้าหากโค้ดไม่มีความผิดพลาด โปรแกรมก็จะทำการคอมไพล์เป็นไฟล์ hex ให้ทันที เพียงแค่นี้เราก็จะได้ไฟล์ hex แล้ว



****

การทำไอซีใหม่ให้มี Bootloader

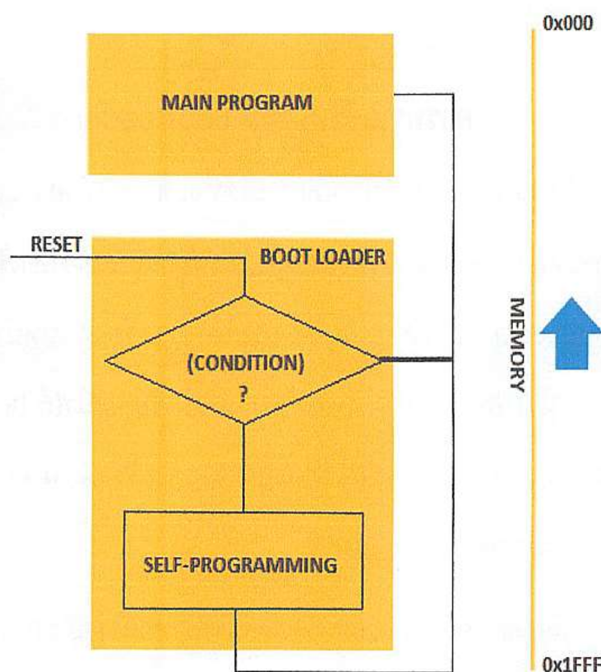
เมื่อเราใช้บอร์ด Arduino เราก็หลีกเลี่ยงไม่ได้ที่จะต้องมาทำความรู้จักกับ Bootloader กันสักหน่อย เพราะว่าการใช้งานที่แสนจะง่ายดายของบอร์ดไมโครคอนโทรลเลอร์สำเร็จรูป Arduino นั้นจะเกิดขึ้นไม่ได้หากว่าไม่มีเจ้า Bootloader การใช้งานไมโครคอนโทรลเลอร์ยุคก่อนหน้านี้สิ่งที่จะขาดไม่ได้คือเครื่องโปรแกรมที่ต้องใช้ในการนำไฟล์โปรแกรมที่คอมไพล์แล้วที่เรียกว่าไฟล์ hex เพื่อเขียนข้อมูลนั้นเข้าสู่หน่วยความจำหลักของตัวไมโครคอนโทรลเลอร์ เพื่อให้ทำงานตามโปรแกรมที่เขาเขียนสั่งงานได้ จึงเป็นการยุ่งยากพอสมควรที่จะใช้งานไมโครคอนโทรลเลอร์ได้

Bootloader คือชุดคำสั่งโปรแกรมขนาดเล็กที่ใช้จัดการสั่งงานในการเริ่มต้นของการทำงาน มันจะเป็นตัวจัดการควบคุมฮาร์ดแวร์ที่อยู่ภายใน ตามโปรแกรมของ Bootloader นั้น ไมโครคอนโทรลเลอร์ตระกูล AVR ATmega เกือบทั้งหมดมีความสามารถในการตั้งโปรแกรมให้กับ Bootloader และ Bootloader สามารถมีได้หลายขนาด และสามารถใช้การเชื่อมต่อที่จะอัพโหลดโปรแกรมได้แตกต่างกัน เช่น UART, SPI, I2C, USB

ที่อยู่เริ่มต้นของโปรแกรมนั้น ขึ้นอยู่กับการตั้งค่า Fuse ของ AVR ต้องดูจากดาต้าชีท เช่นถ้าการตั้งค่า Fuse สั่งให้โปรแกรมตัวนับเข้าสู่ส่วนของ Bootloader หลังจากดำเนินการรีเซ็ตแล้ว ก่อนอื่นจะต้องรันโปรแกรม Bootloader ก่อนที่จะเข้าทำงานในโปรแกรมหลัก ถ้าไมโครคอนโทรลเลอร์ตั้งค่าไว้ล่วงหน้าแล้ว หลังจากทำการรีเซ็ต จะเริ่มทำงานจากตำแหน่งเริ่มต้นของหน่วยความจำเริ่มต้นไม่ได้ (โดยปกติจะอยู่ที่ตำแหน่ง 0 × 0000) แต่ในตำแหน่งเฉพาะบางแห่งโดยปกติจะเป็นพื้นที่ของ Bootloader แต่ในการ Burn Bootloader ด้วย Arduino นั้นผู้ใช้ไม่จำเป็นต้องมีความรู้เรื่องการ Fuse bit เพราะทุกอย่าง Arduino จะเป็นตัวจัดการให้

เองโดยอัตโนมัติ แม้แต่ไฟล์ Bootloader เอง Arduino จะเป็นตัวเข้าไปเลือกเองตามเมนูที่เราได้เลือกไว้ว่าใช้บอร์ดรุ่นใด จึงไม่ต้องเป็นห่วงเรื่องนี้ สบายใจได้

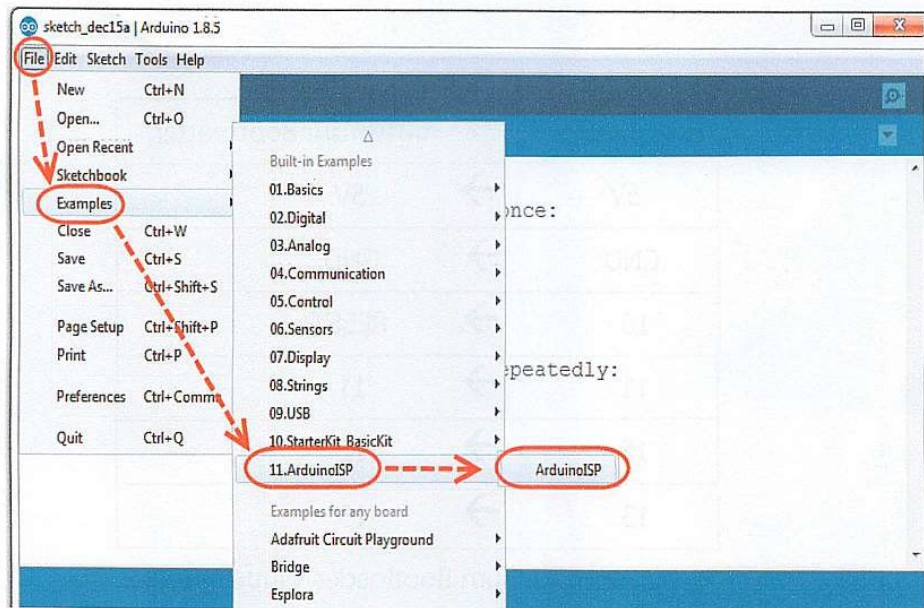
การที่จะทราบได้ว่า Bootloader เสียหายนั้น หากไม่สามารถอัพโหลดโปรแกรมลงบอร์ดได้ในเบื้องต้นให้สังเกตจากหลอด LED_BUILTIN ที่อยู่ในบอร์ด Arduino หาก Bootloader ยังดีอยู่ LED จะติดกระพริบในตอนจ่ายไฟเลี้ยงเข้าบอร์ดในตอนแรก เป็นการส่งสัญญาณบอกว่าโปรแกรมหลักได้เริ่มทำงานได้แล้ว แต่หาก LED ไม่ติดกระพริบแสดงว่า Bootloader อาจจะเสียหาย หรือตัวไอซีเสียเองก็เป็นได้ หรือไม่เช่นนั้นก็อาจจะมีอุปกรณ์อื่นๆในบอร์ดเสียเอง



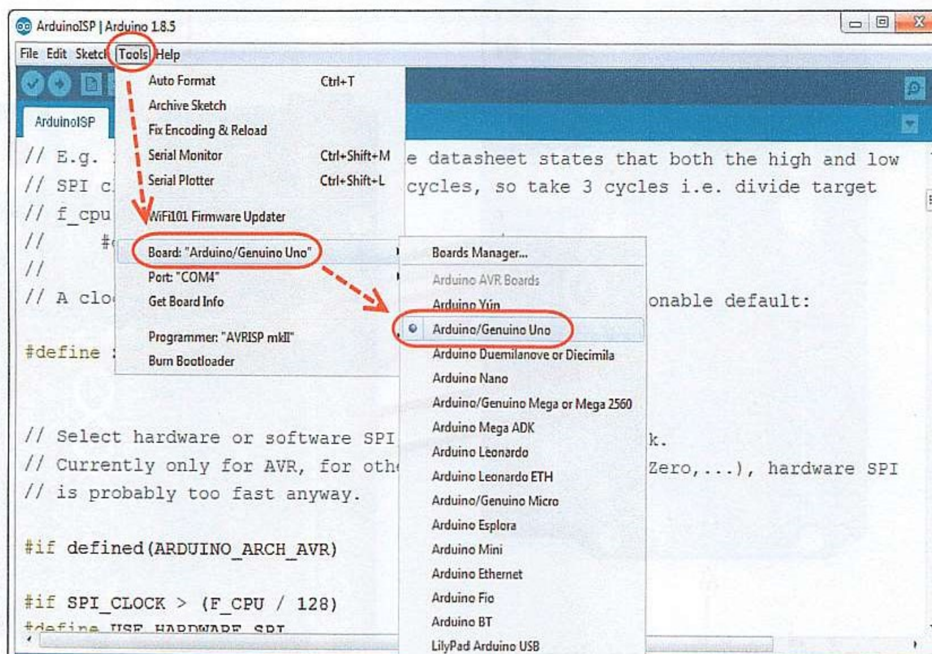
หากจะสรุปให้เข้าใจได้ง่ายๆก็คือ Bootloader ใน Arduino นั้นทำหน้าที่ช่วยในการอัพโหลดโปรแกรมที่เขียนขึ้นมาจาก Arduino IDE เข้าสู่บอร์ด Arduino โดยผ่านทางพอร์ต USB ได้ทันที โดยไม่ต้องใช้เครื่องโปรแกรม ดังนั้นหากเกิดความเสียหายกับ Bootloader แล้วเราก็จะไม่สามารถอัพโหลดโปรแกรมลงบอร์ดได้ หรือ ถ้าหากตัวไอซีเกิดความเสียหาย แล้วเราไปซื้อไอซีตัวใหม่มาใส่ เราก็จะไม่สามารถอัพโหลดโปรแกรมลงบอร์ดโดยผ่านช่องทาง USB ได้ และแน่นอนว่าไอซีตัวใหม่จากโรงงานผู้ผลิตนั้น ไม่มีโปรแกรม Bootloader มาให้ หากเราจะเอาไอซีมาใช้กับ Arduino จะต้องทำการโปรแกรมส่วนของ Bootloader เสียก่อนจึงจะใช้งานเป็น Arduino ได้ และโปรแกรม Arduino IDE สามารถที่จะ Burn Bootloader ได้ด้วยวิธีง่ายๆ แต่จะต้องทำการเบิร์นผ่านเครื่องโปรแกรมของ AVR เท่านั้น แต่ถ้าหากไม่มีเครื่องโปรแกรม เราก็ยัง

สามารถใช้บอร์ด Arduino ที่ยังใช้งานได้อยู่ นำมาสร้างเป็นเครื่องโปรแกรมได้โดยในที่นี้จะยกตัวอย่างการใช้บอร์ด UNO R3 มาทำเป็นเครื่องโปรแกรมด้วยขั้นตอนที่ง่ายมาก

ขั้นตอนการทำนั้น ให้คลิกที่เมนู File → Examples → ArduinoISP → ArduinoISP



คลิกที่เมนู Tools เลือกบอร์ด Arduino/Genuino Uno แล้วทำการอัพโหลดโปรแกรมลงบอร์ด

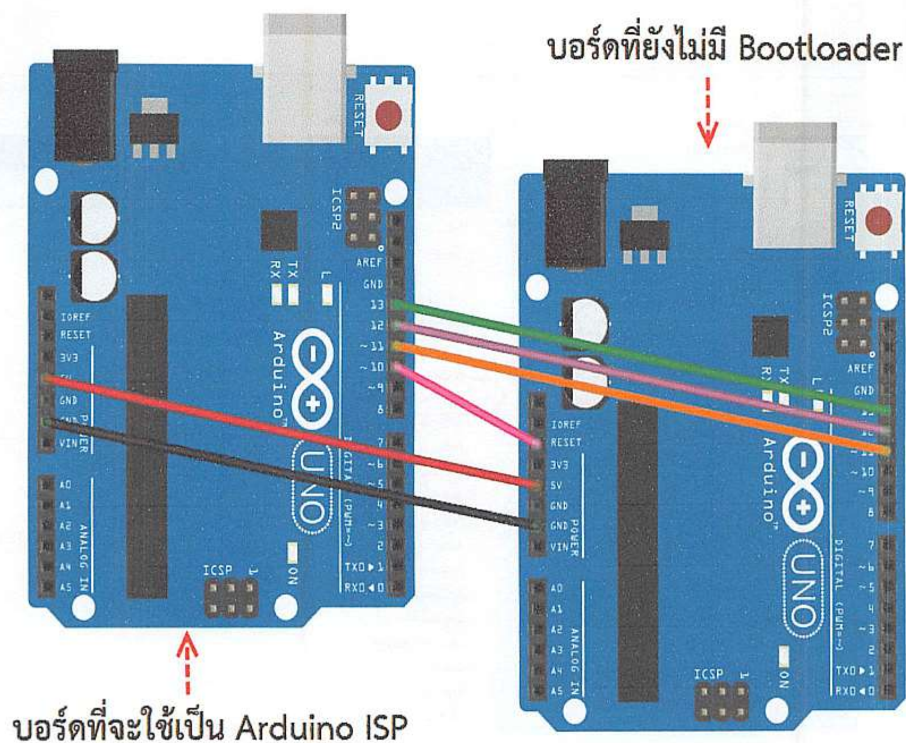


เมื่อทำการอัพโหลดเสร็จแล้วบอร์ด Arduino บอร์ดนี้ ก็จะกลายเป็นเครื่องโปรแกรม Arduino ISP

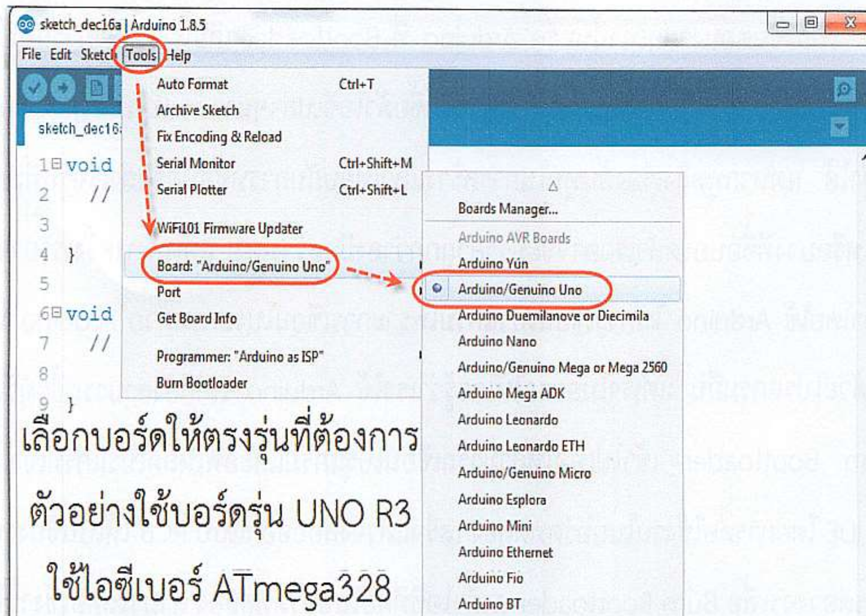
การ Burn Bootloader นั้นสามารถทำได้สองแบบ คือ ทั้งการเชื่อมต่อโดยตรงกับบอร์ด Arduino ที่ Bootloader เสียหาย และเบิร์นเข้าสู่ตัวไอซีเปล่าๆได้ โดยต่อบนบอร์ดทดลอง และขั้นตอนการต่อสายเพื่อทำการเบิร์นนั้นทำได้ง่ายๆโดยการต่อสายไฟเพียง 6 เส้นเท่านั้น

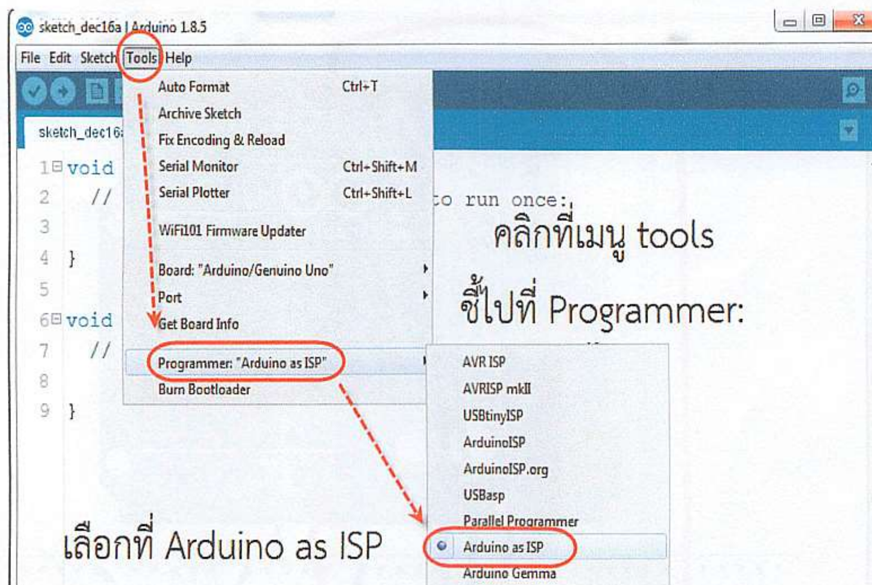
ArduinoISP	→	บอร์ดที่ไม่มี Bootloader
5V	→	5V
GND	→	GND
10	→	RESET
11	→	11
12	→	12
13	→	13

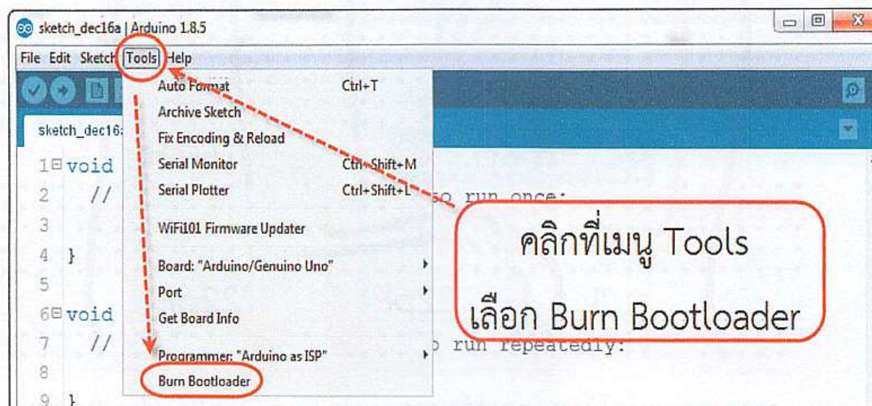
การต่อสายไฟเพื่อ Burn Bootloader จากบอร์ดสู่บอร์ด



เมื่อต่อสายไฟครบทั้ง 6 เส้นและตรวจสอบให้แน่ใจว่าถูกต้องแล้วก็จะเป็นขั้นตอนการ Burn Bootloader

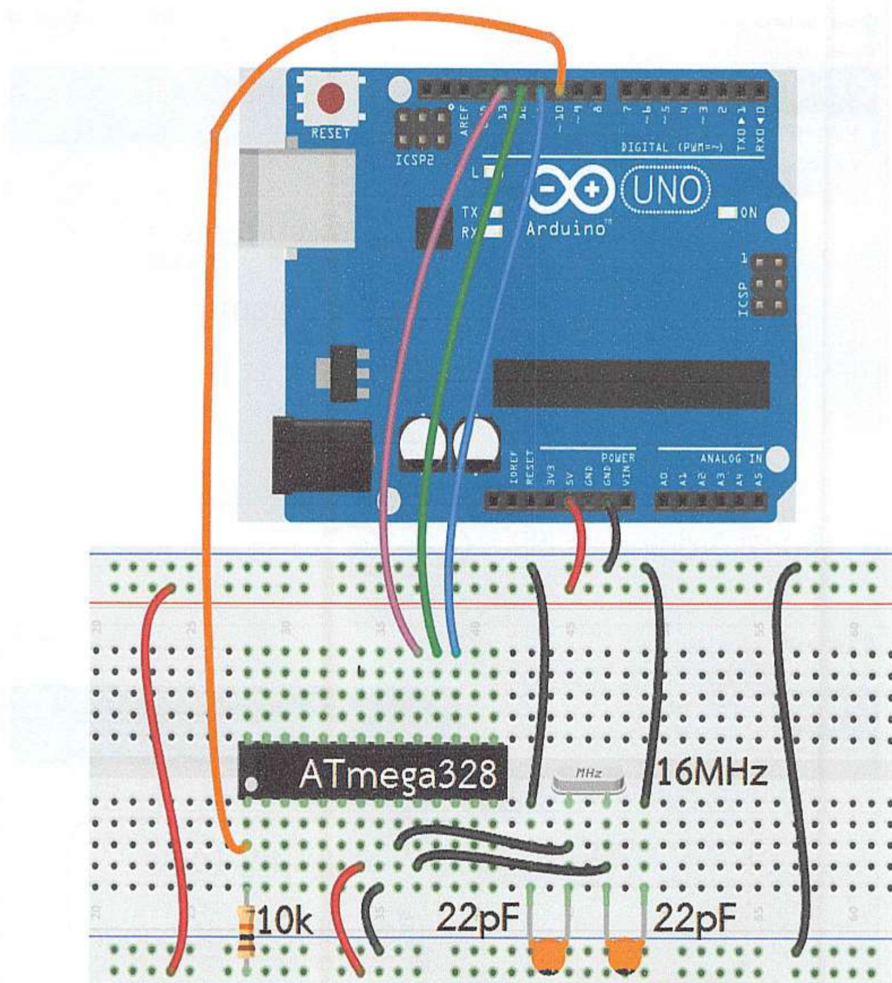




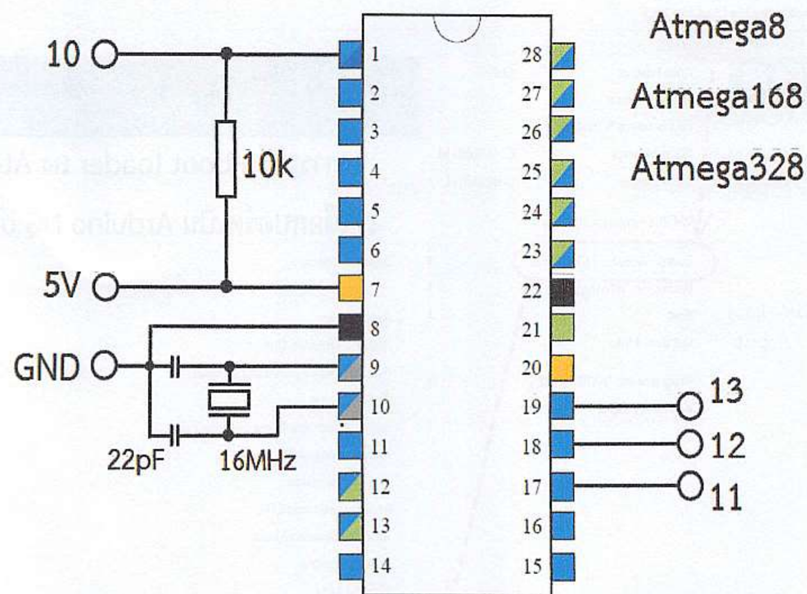


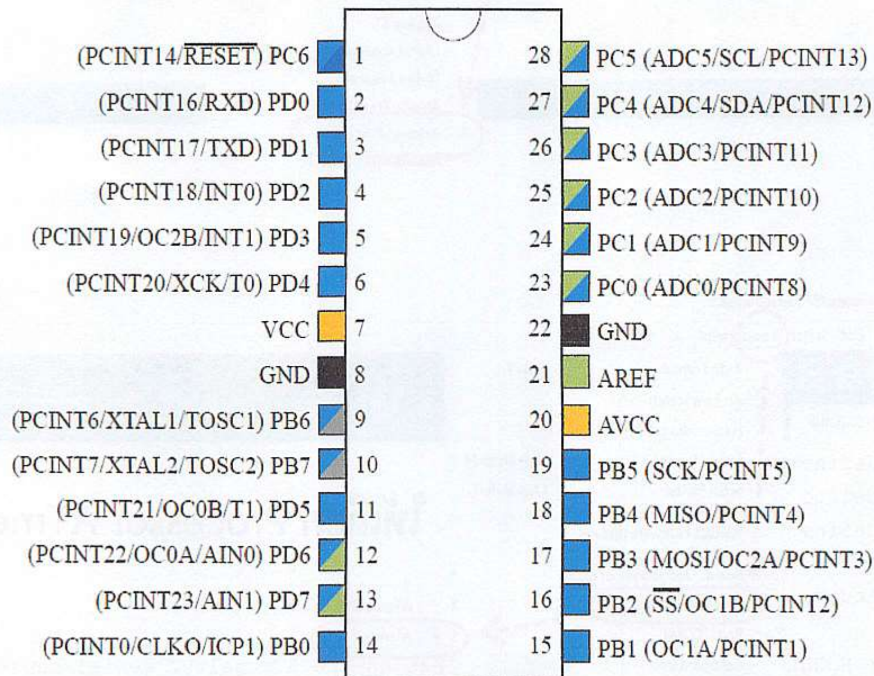
เมื่อเสร็จแล้วที่แถบสถานะด้านล่างจะแสดงข้อความว่า **Done burning bootloader**

เพียงขั้นตอนง่ายๆเท่านี้บอร์ด Arduino ที่ Bootloader เสียหายไม่สามารถอัพโหลดโปรแกรมลงบอร์ดได้ ก็จะกลับมาใช้งานได้ดังเดิม หรือเราอาจจะซื้อตัวไอซีเปล่าๆมาแล้วสร้างบอร์ด Arduino ขึ้นมาใช้เองก็สามารถทำได้ แต่หากพูดถึงเรื่องต้นทุนในการสร้างแล้วเทียบกับการที่ซื้อบอร์ดสำเร็จรูปที่เขาทำมาขายราคาก็พอๆกัน หรือบางทีซื้อบอร์ดสำเร็จอาจจะมีราคาถูกกว่าลงมือสร้างเอง แต่ในบางครั้งเราอาจต้องการสร้างงานอะไรขึ้นมาโดยใช้ Arduino ในการเขียนโปรแกรมเพราะการเขียนโปรแกรมด้วย Arduino เป็นเรื่องที่ง่ายกว่าการเขียนด้วยโปรแกรมอื่น แต่เราไม่อยากให้ใครรู้ว่าเราใช้ Arduino เราก็ยังสามารถซื้อตัวไอซีเปล่าเอามาทำการ Burn Bootloader เข้าไปเพื่อให้สามารถเขียนโปรแกรมและอัพโหลดโปรแกรมเข้าไปในตัวไอซีได้ด้วย Arduino IDE โดยการต่อใช้งานในบอร์ดทดลอง เสร็จแล้วจึงค่อยออกแบบ PCB ให้เป็นชิ้นงานตามแบบของเราเองได้ การต่อวงจรเพื่อ Burn Bootloader ลงตัวไอซีที่ต่อในบอร์ดทดลอง สามารถต่อใช้งานได้ตามรูป



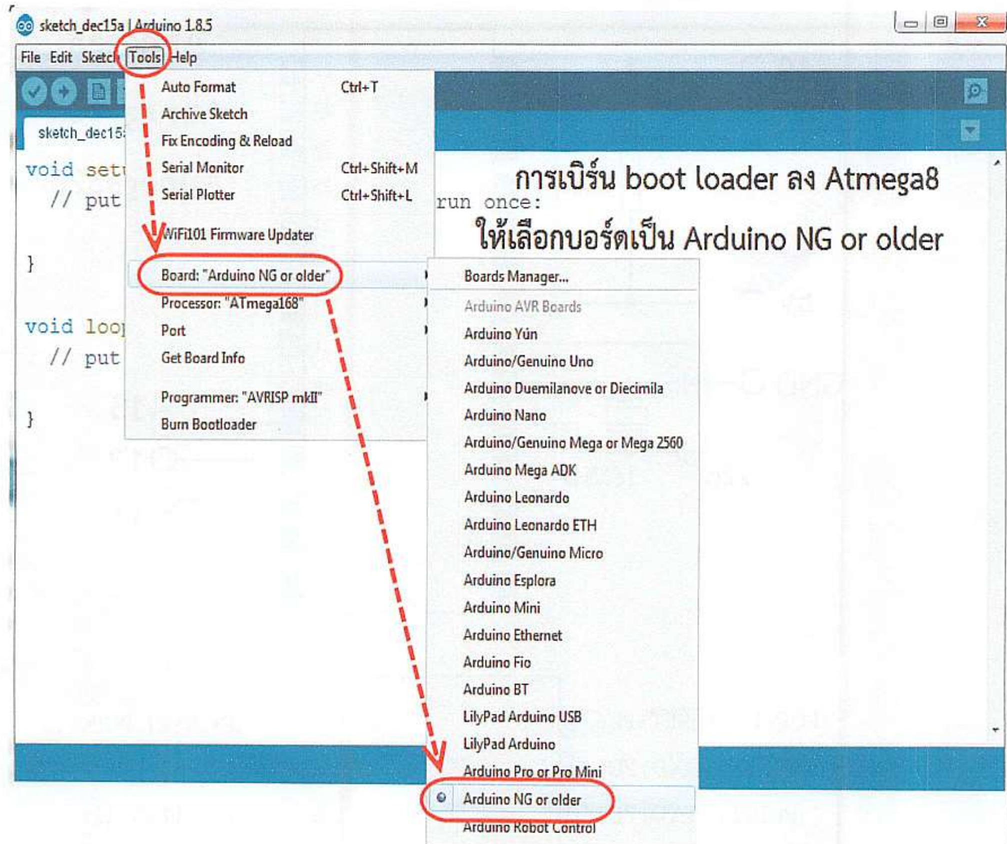
ภาพวงจรการต่อใช้งานบนบอร์ดทดลองเพื่อ Burn Bootloader แบบชัดๆ

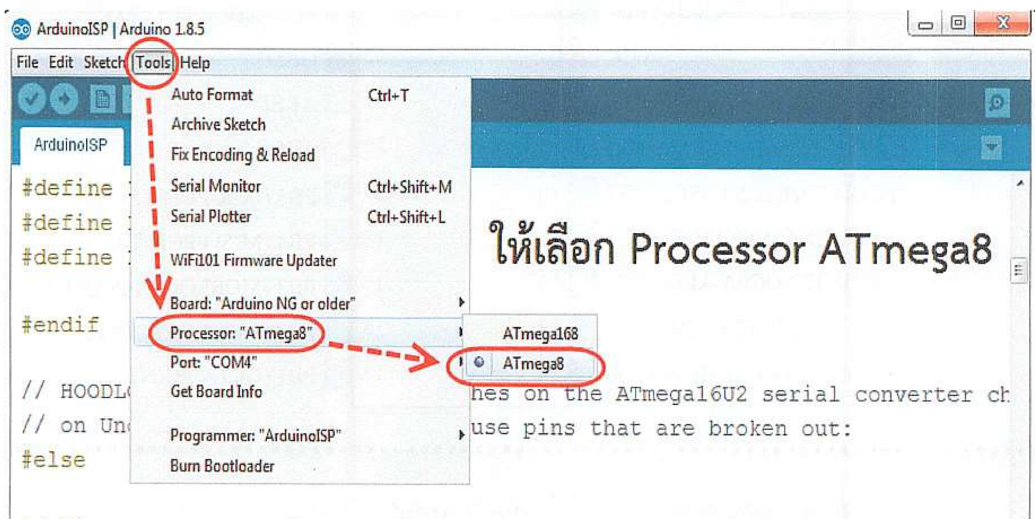


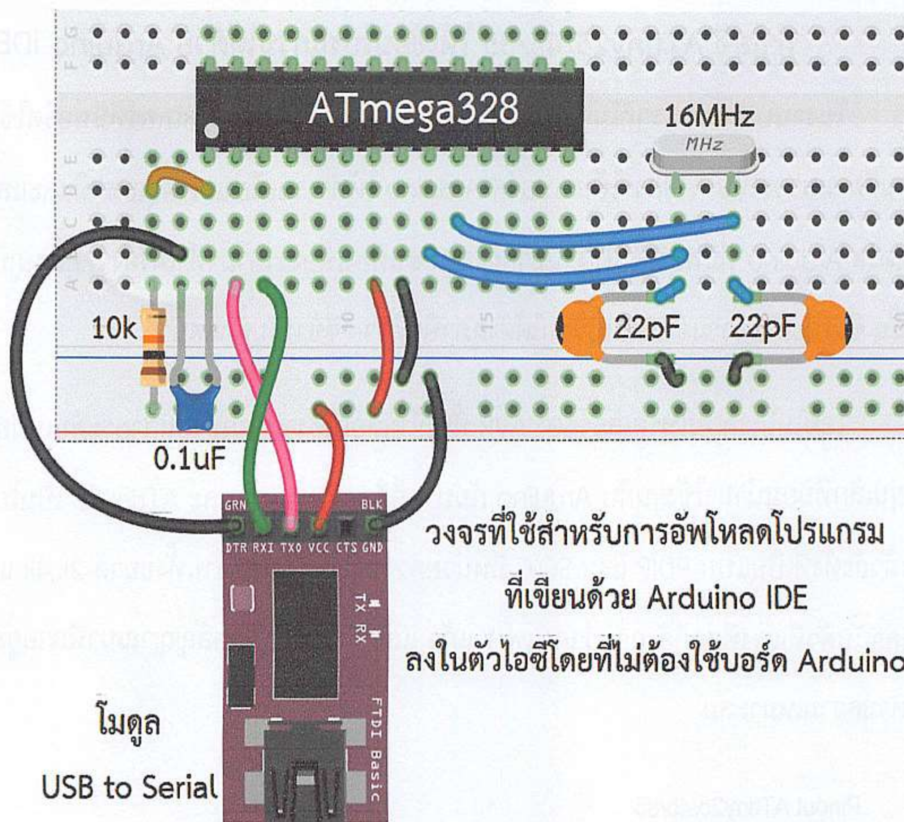


ภาพตำแหน่งขาต่างๆของตัวไอซี ATmega8/ATmega168/ATmega328

สำหรับการ Burn Bootloader ไอซีเบอร์ Atmega8 นั้นจะใช้การต่อวงจรในการเบิร์นเหมือนกัน แต่จะมีความแตกต่างกันที่ขั้นตอนการตั้งค่าในการเลือกบอร์ดที่จะทำการ Burn Bootloader ให้เลือก Board: ไปที่ Arduino NG or older แล้วเลือก Processor เป็น ATmega8 เท่านั้นเอง


sketch_dec15a | Arduino 1.8.5
File Edit Sketch Tools Help
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 Firmware Updater
Board: "Arduino NG or older"
Processor: "ATmega168"
Port
Get Board Info
Programmer: "AVRISP mkII"
Burn Bootloader
การเบิร์น boot loader ลง Atmega8
ให้เลือกบอร์ดเป็น Arduino NG or older
Boards Manager...
Arduino AVR Boards
Arduino Yún
Arduino/Genuino Uno
Arduino Duemilanove or Diecimila
Arduino Nano
Arduino/Genuino Mega or Mega 2560
Arduino Mega ADK
Arduino Leonardo
Arduino Leonardo ETH
Arduino/Genuino Micro
Arduino Esplora
Arduino Mini
Arduino Ethernet
Arduino Fio
Arduino BT
LilyPad Arduino USB
LilyPad Arduino
Arduino Pro or Pro Mini
Arduino NG or older
Arduino Robot Control


ArduinoISP | Arduino 1.8.5
File Edit Sketch Tools Help
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 Firmware Updater
Board: "Arduino NG or older"
Processor: "ATmega8"
Port: "COM4"
Get Board Info
Programmer: "ArduinoISP"
Burn Bootloader
ให้เลือก Processor ATmega8
ATmega168
ATmega8



เมื่อทำการ Burn Bootloader เสร็จแล้วหากต้องการใช้ไอซีทดลองในบอร์ดทดลองสามารถทำได้โดยการใช้บอร์ด USB to Serial เข้ามาช่วยในการเชื่อมต่อเข้ากับพอร์ต USB ของคอมพิวเตอร์เพื่อใช้ในการอัพโหลดโปรแกรมลงตัวไอซี และในส่วนของวงจรที่ต่อนั้นต้องต่อชุดคริสตอลผลิตความถี่รวมทั้ง R ค่า 10k ที่ต้องต่อกับขา RESET และ C ค่า 0.1uF ต่อระหว่างขา DTR ของโมดูล USB to Serial กับขา RESET ของไอซี ATmega328 และสิ่งสำคัญที่จะลืมไม่ได้ คือต้องเลือกโหมด Programmer กลับไปที่ AVR ISP หรือ AVRISP mkII

และหากต้องการใช้งาน ATmega8 ให้เลือก Board ไปที่ Arduino NG or older หรือหากจะใช้ ATmega168 หรือ ATmega328 ให้เลือก Board ไปที่ Arduino mini หรือ Arduino/Genuino Uno

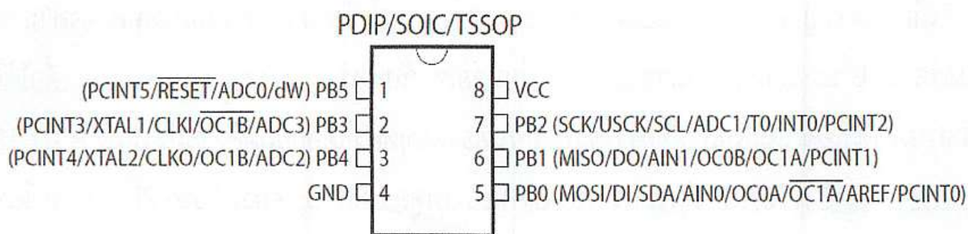
จากการทดลองของผู้เขียนเองในส่วนของ ATmega8 นั้นหากนำมา Burn Bootloader เพื่อใช้งานเป็น Arduino แล้วการทำงานในตอนเริ่มต้นค่อนข้างที่จะบู๊ตช้ามากเป็นเวลาหลายวินาที นี่อาจจะเป็นเหตุผลหนึ่งที่ทำให้ไม่ค่อยได้รับความนิยมมากนักในกลุ่มผู้ใช้งาน Arduino. ดังนั้นหากต้องการใช้งาน ATMEGA8 ให้เร็วนั้นจึงไม่ต้อง Burn Bootloader แต่ให้ complie เป็นไฟล์ hex แล้วทำการเบิร์นผ่าน Arduino ISP เลย ถึงแม้จะเป็นวิธีที่ไม่สะดวกนักแต่จะทำให้ ATMEGA8 บู๊ตได้เร็วกว่า

ทำไอซี ATtiny25/45/85 ให้เขียนโปรแกรมได้ด้วย Arduino IDE

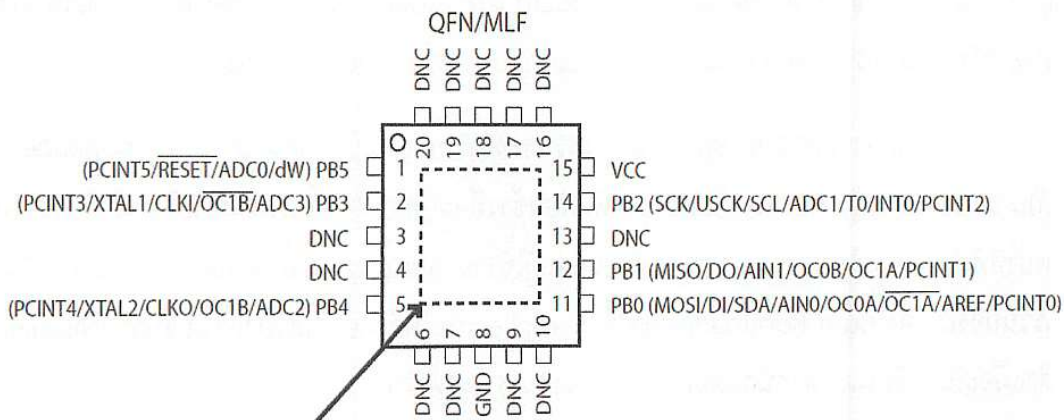
ในงานบางอย่างเราก็ไม่จำเป็นต้องใช้งานไอซีไมโครคอนโทรลเลอร์ที่มีพอร์ตใช้งานมากนัก จากโครงงานที่ผ่านมาจะเห็นได้ว่าการใช้งานบอร์ด Arduino ให้ทำงานเพียงแค่วัดอุณหภูมิและแสดงผลเท่านั้น เราจะซื้อบอร์ด Arduino ที่มีพอร์ตให้ใช้งานมากถึง 20 ขาเพียงเพื่อจะเอามาทำเครื่องวัดอุณหภูมิ ที่ใช้งานเพียง 3 ขาเท่านั้น ซึ่งดูจะเกินความจำเป็น เปรียบได้ดังสุภาษิตที่ว่า “ขี่ช้างจับตั๊กแตน”

ไอซีไมโครคอนโทรลเลอร์ของ AVR นั้นมีให้เลือกใช้งานมากมายหลายระดับเหมือนกับตระกูลอื่นๆ ซึ่งไอซีรุ่นเล็กที่นิยมนำมาใช้งานกับ Arduino กันมากก็คือ ATtiny45 และ ATtiny85 เป็นไอซีที่มีขาใช้งาน 8 ขาและตัวถังทั้งที่เป็นแบบ PDIP และ SOIC มีหน่วยความจำให้เลือกใช้งานทั้งขนาด 2k,4k และ 8k ตามเบอร์ ส่วนราคานั้นตัวที่แพงที่สุดก็ยังถูกกว่ากาแฟ 1 แก้ว และยังมีหน่วยผลิตสัญญาณนาฬิกาอยู่ภายในให้เลือกใช้งานได้ตามความเหมาะสม

Pinout ATtiny25/45/85



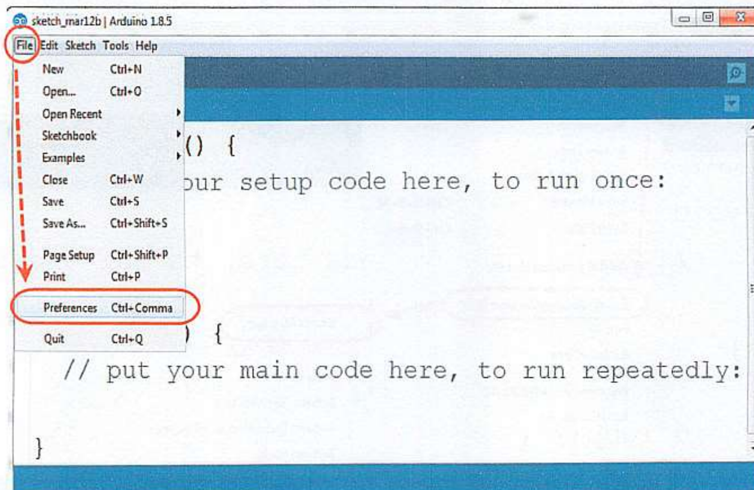
NOTE: TSSOP only for ATtiny45/V



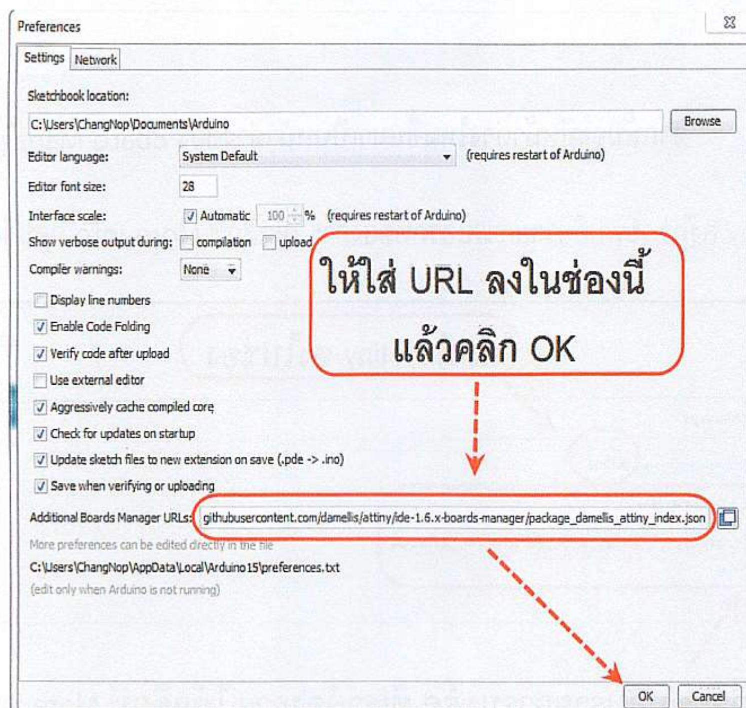
NOTE: Bottom pad should be soldered to ground.
DNC: Do Not Connect

แต่ว่าค่ามาตรฐานของโปรแกรม Arduino IDE นั้นไม่ได้ออกแบบมาให้สามารถเขียนโปรแกรมควบคุมไอซี ATtiny ได้ หากต้องการจะเขียนโปรแกรมสั่งงาน ATtiny25/45/85 ด้วย Arduino จะต้องทำการเพิ่มเบอร์ไอซี ATtiny เข้ามาในโปรแกรมเสียก่อน โดยทำตามขั้นตอนต่อไปนี้

1. ให้คลิกที่เมนู File แล้วเลือก Preferences

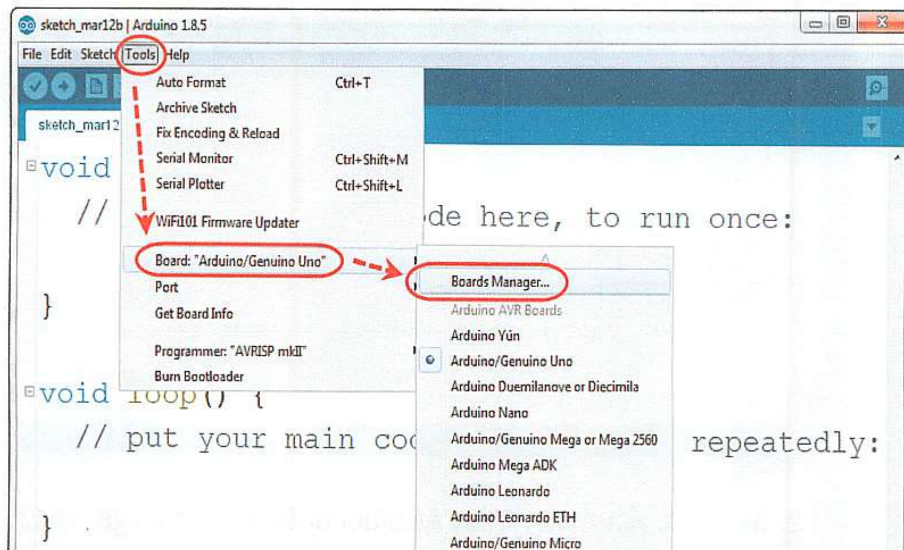


2. ให้ใส่ URL ด้านล่างลงในช่อง Additional Boards manager URLs:



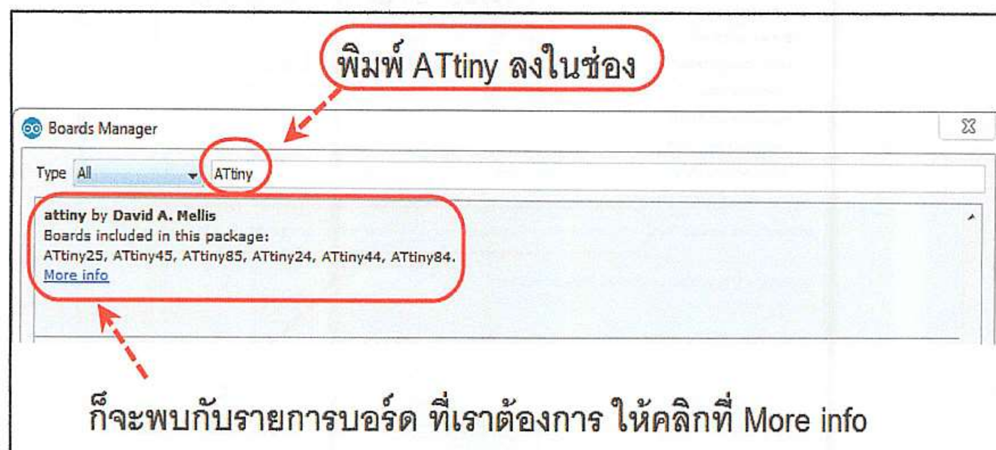
https://raw.githubusercontent.com/damellis/attiny/ide-1.6.x-boards-manager/package_damellis_attiny_index.json

ต้องทำความเข้าใจกันสักนิด ว่าในส่วนของ URL ตรงนี้เป็นช่องทางในการค้นหาเพื่อดาวน์โหลดรายการบอร์ดต่าง ๆ ที่ ค่ามาตรฐานของ Arduino ไม่มีมาให้ เช่นบอร์ด WIFI ESP8266 หรือ STM32 ก็จะสามารถเพิ่มเข้ามาได้ด้วยการใช้วิธีเดียวกันนี้ แต่จะต่างกันที่ URL ที่จะใช้ในการดาวน์โหลดนั้นจะไม่เหมือนกัน ในที่นี้จะยังไม่กล่าวถึง จะขอยกตัวอย่างเพียง ATtiny25/45/85 เท่านั้น เมื่อใส่ URL แล้วกดปุ่ม OK จากนั้นให้คลิกที่เมนู Tools → Board: xxx... → Board Manager ตามรูปตัวอย่าง

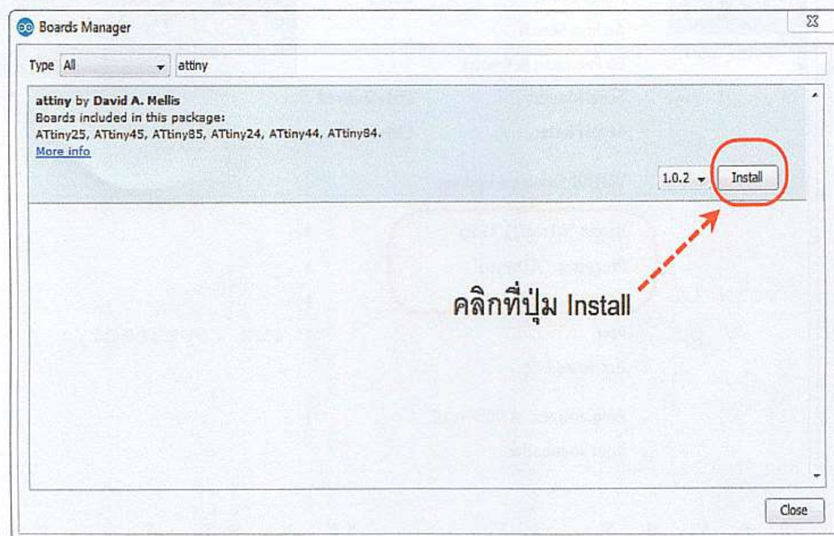


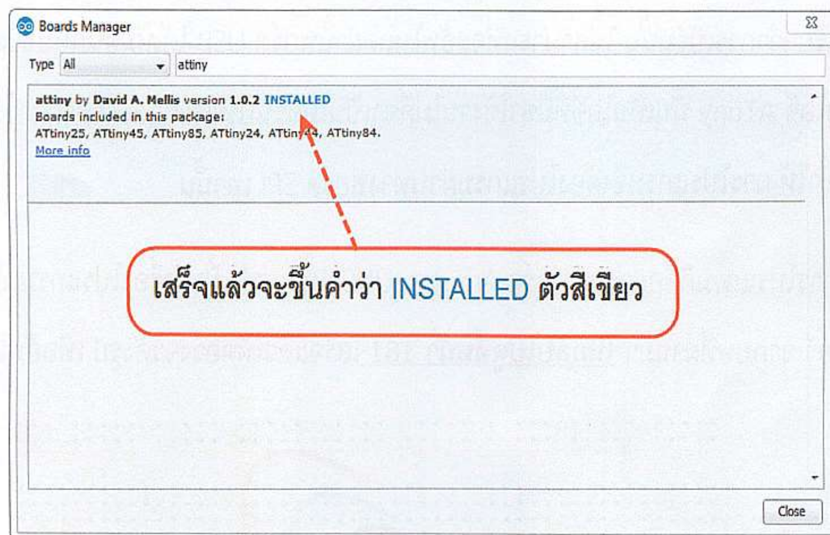
จากนั้นจะมีหน้าต่างใหม่ขึ้นมาเป็นหน้าต่างของ Board Manager

ให้พิมพ์ ATtiny ลงไป จะพบกับรายการบอร์ดที่ต้องการ ให้คลิกที่ More info ก่อนจึงจะพบกับปุ่ม Install

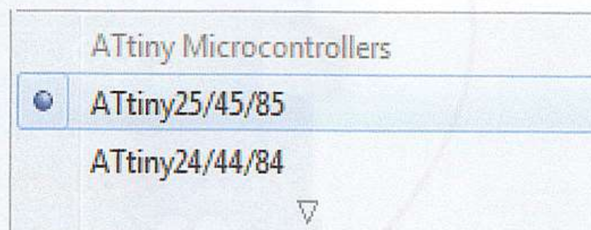


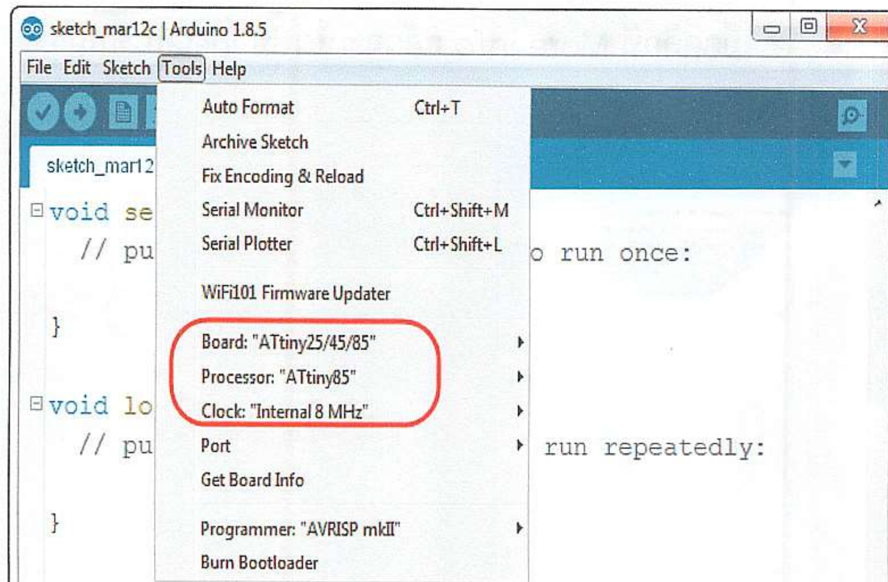
เมื่อคลิกที่ More info แล้วจะเห็นปุ่ม Install ขึ้นมา





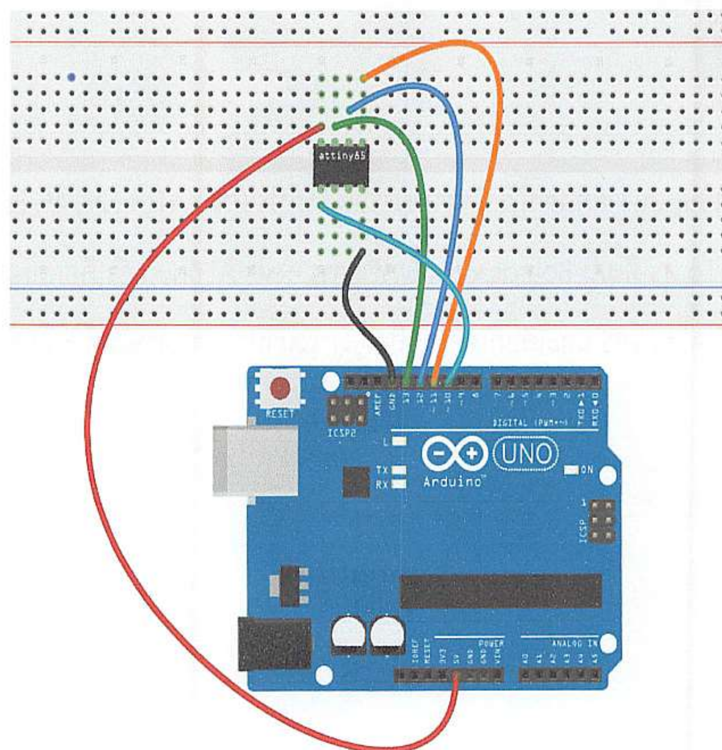
เพียงขั้นตอนง่ายๆเท่านี้เราก็จะสามารถเขียนโปรแกรม ATtiny25/45/85 ด้วย Arduino ได้แล้ว โดยการไปที่เมนู Tools ที่เมนู Boards:xxx แล้วทำการมองหา ATtiny แล้วเลือกเบอร์ที่ต้องการใช้งาน ในที่นี้ผู้เขียนจะใช้เบอร์ ATtiny85 และเลือกใช้ความถี่สัญญาณนาฬิกาจากภายในที่ 8MHz





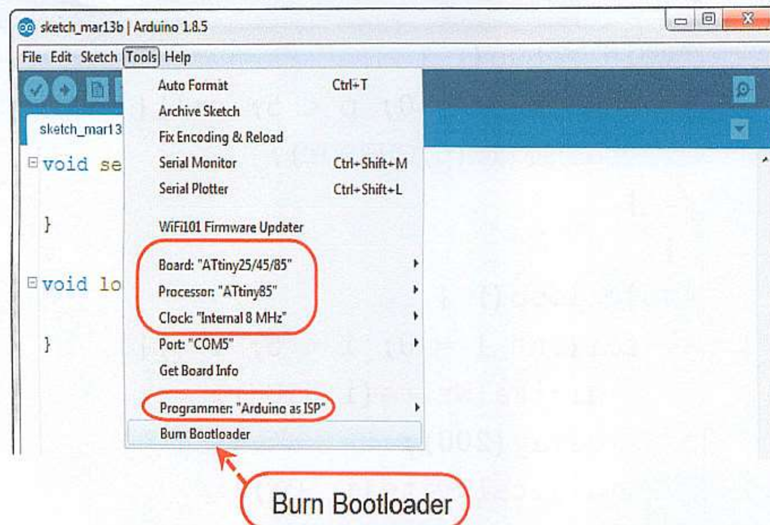
แต่ว่า ถึงแม้จะเขียนโปรแกรมด้วย Arduino ได้ แต่การอัพโหลดโปรแกรมที่เขียนขึ้นมานั้นเข้าสู่ตัวไอซี หรือที่เรียกว่าการเบิร์นนั้น ไม่สามารถที่จะอัพโหลดผ่านพอร์ต USB ได้เหมือนกับบอร์ด Arduino ทั่วไป เพราะว่าตัวไอซี ATtiny นั้นเป็นไอซีที่มีขาใช้งานไม่กี่ขาเป็นไอซีรุ่นประหยัดและราคาถูก จึงไม่มีพอร์ต USART หรือ USB มาให้ การโปรแกรมจึงต้องโปรแกรมผ่านทางพอร์ต SPI เท่านั้น

วิธีที่ง่ายในการโปรแกรมก็คือการเอาบอร์ด Arduino UNO R3 มาทำเป็นเครื่องโปรแกรมเช่นเดียวกับการเบิร์นบู๊ตโหลดเดอร์ จากบทที่ผ่านมา **ให้กลับไปดูที่หน้า 181** เสร็จแล้วให้ต่อวงจรดังรูป เพื่ออัพโหลดโปรแกรม



สำหรับค่าสัญญาณนาฬิกาจากภายในของ ATtiny85 นั้นโดยปกติจะทำงานที่ความถี่ 1MHz ถ้าหากต้องการสัญญาณนาฬิกาที่มากกว่า 1MHz เช่น 8MHz ต้องทำการ Burn Bootloader ก่อน

ในส่วนของ Programmer: ให้เลือกเป็น Arduino as ISP



เมื่อทำการต่อสายทุกอย่างถูกต้องและทำตามขั้นตอนทุกอย่างอย่างถูกต้องแล้ว ก็ทดลองเขียนโปรแกรมได้ แต่ต้องทำความเข้าใจในเรื่องของขาที่จะใช้งานก่อน เพราะ ATtiny25/45/85 นั้น มีพอร์ต IO หรือขา ให้ใช้งานได้ทั้งหมด 5 ขาเท่านั้น

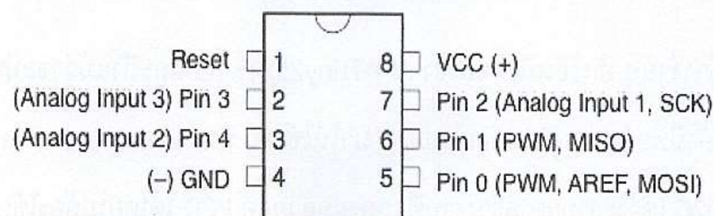
ขา 5 ของไอซี = 0

ขา 6 ของไอซี = 1

ขา 7 ของไอซี = 2

ขา 2 ของไอซี = 3

ขา 3 ของไอซี = 4



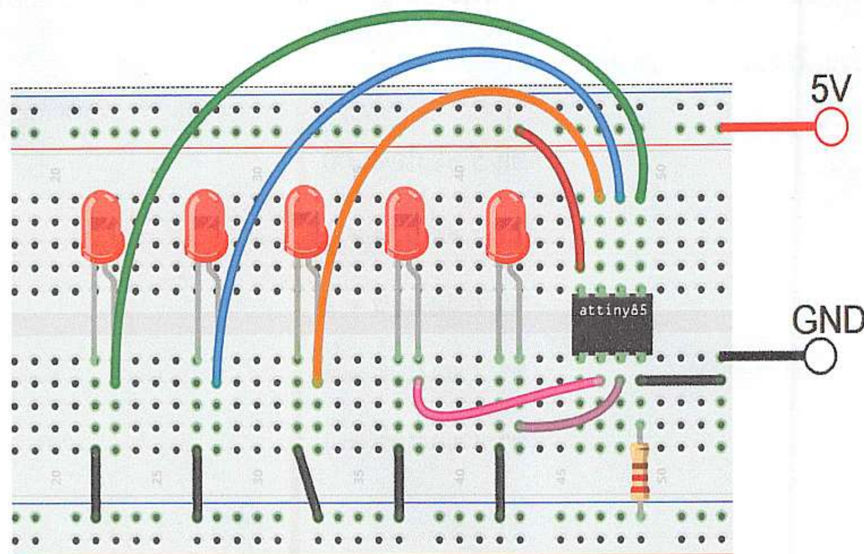
สำหรับตัวอย่างโปรแกรมทดลองนี้เราจะใช้วงจรไฟวิ่ง 5 ดวง เป็นการใช้งานขาไอซีทั้งหมดมาใช้งานเป็นเอาต์พุตทั้งหมด เพื่อให้ผู้อ่านได้เห็นตัวอย่างการสั่งงานขาทุกขาของ ATtiny85 อย่างชัดเจน

ATtiny85_test | Arduino 1.8.5

File Edit Sketch Tools Help

ATtiny85_test §

```
void setup() {
  for(int p = 0; p < 5; p++){
    pinMode(p,OUTPUT);
  }
}
void loop() {
  for(int i = 0; i < 5; i++){
    digitalWrite(i,HIGH);
    delay(200);
    digitalWrite(i,LOW);
  }
}
```



การต่อวงจรเพื่อทดลองการทำงานของโปรแกรมวิ่ง 5 ดวง ที่ทำงานด้วย ATtiny85

นี่เป็นเพียงตัวอย่างง่ายๆ อีกวิธีหนึ่งที่จะใช้งาน ATtiny25/45/85 และถึงแม้ว่าจะมีขาใช้งานได้แค่เพียง 5 ขา แต่ ATtiny85 ก็ยังมีพอร์ตการเชื่อมต่อแบบ I2C มาให้ใช้งาน นั่นหมายความว่าเราสามารถที่จะต่อจอแสดงผล LCD16*2 แบบ I2C ได้ จึงสามารถสร้างงานที่แสดงผลด้วยจอ LCD ได้โดยไม่ต้องใช้บอร์ด Arduino เลย

เทคนิคการประกาศขาอินพุต หรือ เอาต์พุต หลายๆขา

เป็นที่ทราบกันดีอยู่แล้วว่าก่อนจะใช้งานพอร์ตหรือขาต่าง ๆ ของ Arduino นั้นจะต้องประกาศก่อนการใช้งานว่าจะใช้ขานั้น ๆ ให้ทำงานเป็นอินพุตเพื่อใช้รับสัญญาณเข้ามา หรือว่าจะใช้ขานั้นเป็นเอาต์พุตเพื่อที่จะส่งสัญญาณออกไปควบคุมอุปกรณ์ต่าง ๆ และการประกาศในเบื้องต้นนั้นต้องประกาศไว้ในฟังก์ชันที่ชื่อว่า void setup() โดยคำสั่งที่ใช้นั้นเป็นฟังก์ชันสำเร็จรูปที่ชื่อว่า pinMode() หากขาที่ต้องการใช้งานมีไม่มากก็ดูเหมือนว่าจะไม่ใช่ปัญหา จากตัวอย่างเป็นการใช้งานขา 12 และ 13 ทำงานเป็นเอาต์พุต

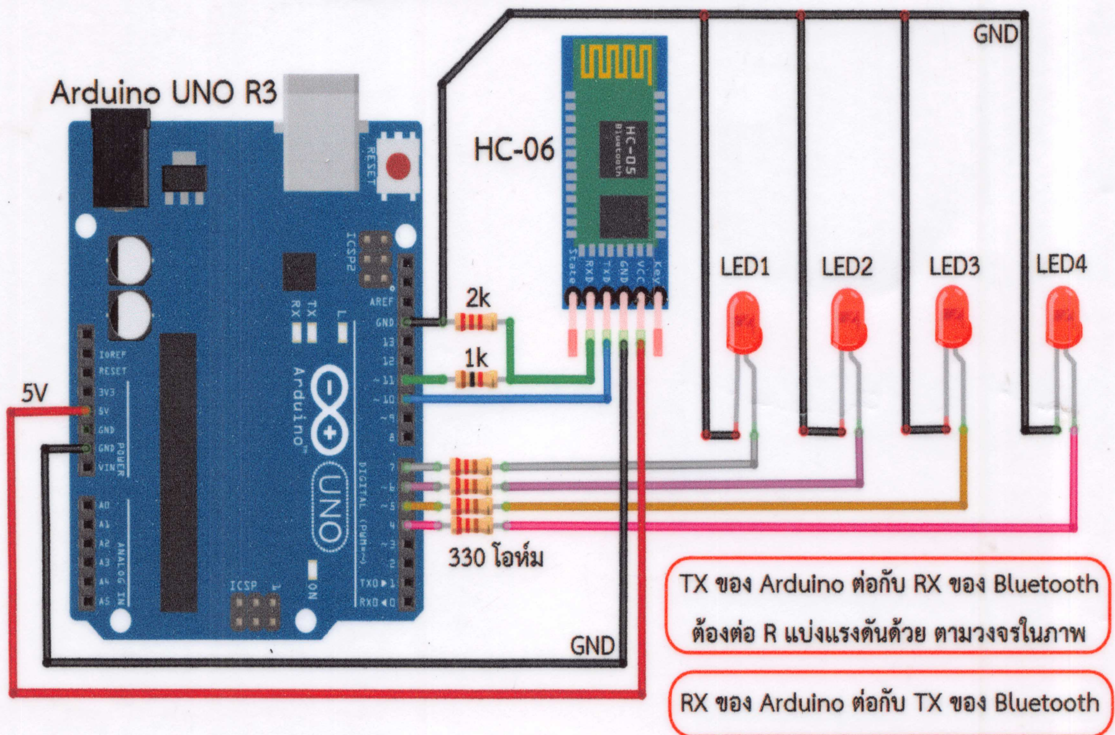
```
void setup() {
  pinMode(12,OUTPUT);
  pinMode(13,OUTPUT);
}
void loop() {
  digitalWrite(12,HIGH);
  digitalWrite(13,HIGH);
}
```

แต่ถ้าหากต้องการใช้งานมากกว่าสิบขา เช่น ต้องการใช้งานพอร์ตดิจิทัลทั้งหมดให้เป็นเอาต์พุตตั้งแต่ขา 0-13 เราจะต้องประกาศ **pinMode()** ทั้งหมดถึง 14 บรรทัด แต่จริง ๆ แล้วเราสามารถใช้งานคำสั่งการทำงานแบบวนซ้ำเช่นคำสั่ง for มาใช้งานได้ทำให้โค้ดที่เขียนขึ้นมานั้นสั้นลงได้อย่างมาก

```
int pins[]={0,1,2,3,4,5,6,7,8,9,10,11,12,13};
void setup() {
  for(int i=0;i<14,i++){
    pinMode(pins[i],OUTPUT);
  }
}
void loop() {
  for(int i=0;i<14;i++){
    digitalWrite(pins[i],HIGH);
  }
}
```

จากตัวอย่างจะเห็นได้ว่าเราสามารถประกาศใช้งานขา 0 ถึง 13 เป็นเอาต์พุตได้ทั้งหมดโดยการเขียนโค้ดเพียงแค่สองบรรทัดเท่านั้นเอง โดยอาศัยการทำงานของลูป for นั่นเอง

อย่างไรก็ดีข้อมูลและเนื้อหาในหนังสือเล่มนี้ถูกจัดทำขึ้นมาในระยะเวลาอันสั้น เพื่อใช้ในการอบรมสอนการเขียนโปรแกรมควบคุม Arduino เบื้องต้นให้กับผู้ที่สนใจต้องการศึกษาการเขียนโปรแกรม Arduino ในระดับผู้เริ่มต้น ดังนั้นข้อมูลต่าง ๆ ในระดับลึก หรือวงจรและโปรแกรมที่ทำงานซับซ้อนจะไม่มีในหนังสือเล่มนี้จุดประสงค์คือเน้นให้ผู้เริ่มต้น ได้ศึกษาและทำตามตัวอย่างได้อย่างไม่ยากเย็นนัก อีกทั้งตัวอย่างโปรแกรมต่างๆ ในหนังสือเล่มนี้ไม่ได้มีความซับซ้อน ผู้เขียนจึงไม่ได้กล่าวถึงเรื่องการเขียน โฟลว์ชาร์ต การทำงานในการเขียนโปรแกรม และจะเขียนถึงเรื่องราวต่าง ๆ ถึงรายละเอียดการทำงานที่ซับซ้อนในระดับลึก ในเล่มต่อไป หากเนื้อหาในหนังสือเล่มนี้มีความผิดพลาดหรือขาดตกบกพร่องประการใด ข้าพเจ้าขอน้อมรับคำติชมและคำแนะนำต่าง ๆ และจะนำไปแก้ไขให้ถูกต้องสมบูรณ์ต่อไป







QR Code สำหรับดาวน์โหลดแอพพลิเคชั่น

www.ornittech.com